JPRS-JST-88-024
343069
13 OCTOBER 1988

FOREIGN BROADCAST INFORMATION SERVICE

JPRS Report

Science & Technology

Japan

COMPUTER ARCHITECTURE

19980612 033

10
65
A04

JPRS-JST-88-024
13 OCTOBER 1988

SCIENCE & TECHNOLOGY
JAPAN
COMPUTER ARCHITECTURE

43067589 Tokyo KONPYUTA AKITEKUCHA SHINPOJIUMU RONBUNSHU in Japanese 17-18 May 88, pp 27-35; 55-64; 89-97; 199-208

[Selected papers from the Computer Architecture Symposium held 17-18 May 1988 in Tokyo, sponsored by the Information Processing Society of Japan]

CONTENTS

Neural Network Simulator Architecture

43067589a Tokyo KONPYUTA AKITEKUCHA SHINPOJIUMU RONBUNSHU in Japanese
17-18 May 88 pp 27-35

[Article by Nobuki Kajiwara, Toshiyuki Nakada, Satoru Matsushita, and Tomohiko Koike of NEC Corp.: "Neural Network Simulation Machine--NeuMan"]

[Text] In recent years, neural networks (NNs) have been actively studied as models for fine-grain high-level parallel computations. Various models and learning algorithms have been proposed and research for the development of neural network hardware has been conducted.

In this paper, a special simulation machine capable of simulating large-scale neural networks at high speed will be discussed. The special simulation machine pipelines the simulation algorithm to be used, divides the target neural network into partial networks, and processes the partial networks in parallel using special processor elements (PEs).

1. Introduction

We have proposed an information- and time-continuous neural network model based on the idea that a neural network is a fine-grain parallel computation model, and have been engaged in the experimental generation of a neural network description language, a compiler and simulators and in experiments on small-scale information processing functions (a simple control system, a forward inference system, time-series pattern recognition, N_Queen, etc.) of a neural network. In testing the neural network, we stored the neural network program as the weight of the links and tested neural network functions by simulation. For the simulations, we used simulators implemented by software on conventional types of computers (LISP machine, VAX, PC9801). The software-based simulation performed on the LISP machine recorded a processing speed of 90 to 400 links per second. (The simulator to be used on the LISP machine has been created without much importance attached to the processing speed. It will be possible to raise its processing speed by about one order of magnitude.) With another experimental simulator created for use on the VAX8650, to which importance has been attached to the processing speed, a processing speed of about 140 K links/s has been achieved. However, even this processing speed will be inadequate for conducting larger-scale experiments or developing practical applications. The inadequate processing speed is attributable to the simulation of parallel computations by consecutive sequence computers. Subsequently, we came to think it necessary to develop a special simulation machine capable of utilizing the parallelism characteristic of neural networks and began to study a special neural network simulation machine--NeuMan. The NeuMan enables high-speed simulation of a large-scale neural network by pipelining the simulation algorithm, dividing the neural network into partial networks and processing the partial networks in parallel using more than one processor element.

2. Basic Design

Two approaches can be taken for the development of a special machine for neural network simulation. In one, analog circuits are used for neural network simulation. In the other, digital circuits are used, as is done in the conventional computers. In an analog circuit used to simulate a neural network, the nodes of the neural network are represented by transistors and operational amplifiers, while the links are represented by resistors. A neural network simulation machine with an analog simulation circuit comprising the elements, as mentioned above, can operate at higher speeds than can a comparable machine comprising a digital circuit.

However, the analog-circuit machine requires the target neural network to be programmed when the analog circuit is formed. Once the neural network is programmed for the machine, it is difficult to modify the program using current techniques. This method is not yet realistic for the simulation of a large-scale neural network. A neural network simulation machine with a digital simulation circuit is slower than a comparable machine with an analog simulation circuit, but it permits program modification and the simulation of a large-scale neural network. If it is designed to be operated under a microprogram, it allows the neural network model to be modified or to be used for learning simply by rewriting the microprogram. We have selected the digital circuit type for its flexibility and capability for large-scale neural network simulation use.

A neural network can be represented by a matrix. Supercomputers and array processors can perform matrix operations at high speeds, so neural network simulators made from those high-speed processors have been proposed. In an analog circuit-type neural network simulator, a neural network is represented by a resistor matrix. The simulation method in which the neural network is represented by a matrix is efficient when the nodes of the neural network are almost perfectly interlinked. However, the storage capacity and hardware quantity required for neural network simulation by this method increases in proportion to the square of the number of nodes. Therefore, with this method, the simulation of a neural network with several thousand or several tens of thousands of nodes is unrealistic.

Generally, each node of a neural network is not linked with all the other nodes of the neural network. The brain of a human is said to comprise several tens of billions of neurons. The number of synapses held by each neuron is said to range from several thousand to several tens of thousands. This is much smaller than the total number of neurons contained in the brain. We previously created a small-scale forward inference system comprised of a neural network. In the neural network, each node had an average of five synapses. When a practical information processing system comprising a neural network is created, the system is divided into function modules (partial networks). In the neural network, although many links exist between nodes within each function module, the number of links between the function modules is not as large. If this type of neural network is represented by a matrix, the matrix becomes sparse, with most of its elements comprising zeroes. Therefore, using a supercomputer, array processor or an analog-circuit machine to simulate the neural network will be inefficient, causing many useless computations to be performed and involving the use of a large quantity of memory and other hardware resources.

Our neural network simulator divides the neural network to be simulated into partial neural networks and extracts their parallelism characteristic by processing them in parallel, using more than one processor. Instead of producing a matrix that would represent all the links in the neural network, a link table (fan-out table)is prepared for each node. It contains information on the links stemming from the node.

Accelerated processing is achieved by pipelining the simulation algorithms, thereby extracting their parallelism characteristic, and also by dividing the neural network into partial networks and processing the partial networks using more than one processor element, thereby extracting their parallelism characteristic.

3. Neural Network Model

Figure 1 illustrates the image of the neural network system proposed here. The "environment" shown in the figure represents where the neural network operates. Generally, it is a dynamic system whose condition changes with time. The neural network consists of the weighted links between each node and many others.

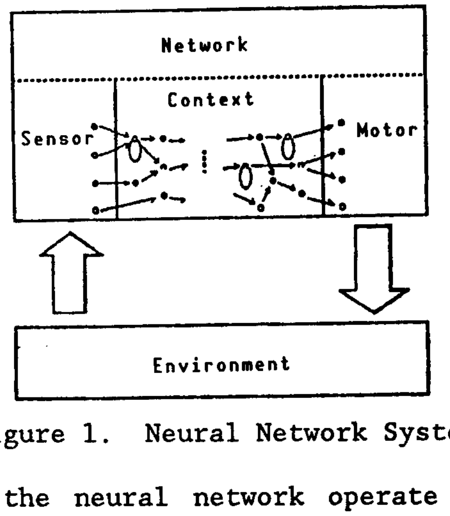


Figure 1. Neural Network System

It is necessary that the neural network operate while recognizing the dynamically changing environment. To meet this need, each node has an internal status, referred to as the activity degree. This internal status changes with time while being affected by the environment and other nodes. The pattern of the activity degrees of the nodes contained in the neural network may be said to represent the current status of the neural network. The neural network is a dynamic system that determines its actions and changes its status according to the current status (the activity degree of each node) and the information obtained from the environment.

The nodes comprise three kinds: sensor, context, and motor nodes. The sensor nodes change their activity degrees according to environmental conditions, while the motor nodes work on the environment according to the activity degrees. The context nodes come between the sensor and motor nodes, realizing the system functions of short-period memorization and information processing.

None of such network structure types as the symmetrical structure or layered structure is assumed for this neural network. An arbitrary structure, which may be formed through the feedbacking of information to itself, may be assumed. For purposes of theoretical analysis or study, it is more convenient to assume a specific network structure. However, from the viewpoint of information processing capacity, it is more advantageous to not assume any specific network structure.

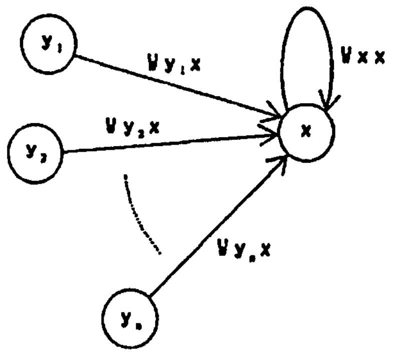


Figure 2. Neural Network Model

We propose the following time- and information-continuous model as a neural network model for recognizing a dynamically changing pattern.[1] Figure 2 shows the neural network model that permits information feedback to itself. The activity degree of node x shown in Figure 2 changes when affected by the activity degrees of other nodes, y_1, y_2, $\cdots$, y_n, and itself. Wy_1x represents the weight of the link between node y_1 and node x. Wxx represents the weight of the link extending from node x to node x itself. When x, y_1, y_2, $\cdots$, and y_n represent the activity degrees of the corresponding nodes, x changes according to the following differential equation:

$$\tau dx/dt = \Sigma Wy_1x \cdot \mu\,(y_1) + Wxx \cdot \mu\,(x) - x \tag{1}$$

where $\Sigma Wy_1x \cdot \mu\,(y_1)$ = influence of other nodes
$Wxx \cdot \mu\,(x)$ = influence of node x itself
x = attenuation term.

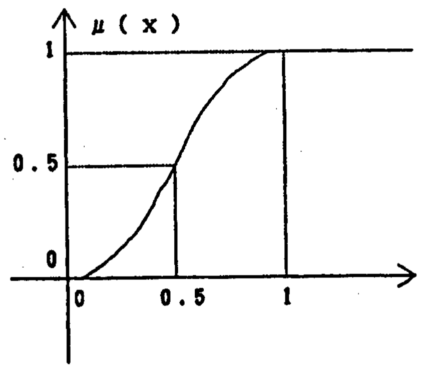


Figure 3. Output Function μ

Equation (1) is called an operation equation. $Wy_1x \cdot \mu(y_1)$ is called the vote from node y_1 to node x. τ is a time constant. The smaller the value of τ, the larger the change in activity degree. the output function μ is a monotonic increase function, in a wide sense, with a limiter characteristic as shown by equation (2) and Figure 3.

$$\mu(x) = \begin{cases} 0 & ;\ x \leq 0 \\ 2x^2 & ;\ 0 < x \leq 0.5 \\ 1-2\ (1-x)^2 & ;\ 0.5\ \ x \leq 1 \\ 1 & ;\ 1 < x \end{cases} \tag{2}$$

In approaches to neural network simulation dependent on digital circuits, including cases where simulation is performed by software running on a general-purpose computer, the operation equation (1) can be solved by Euler's method in succession. The time Δt can be integrated by the following recurrence formula:

$$\begin{aligned} accx &:= \Sigma Wy_1x \cdot \mu(y_1) + Wxx \cdot \mu(x) \\ \Delta x &:= \Delta t\ [accx - x]/\tau \\ x &:= x + \Delta x \end{aligned} \tag{3}$$

The operation of the neural network can be defined by a system of operation equations (1) defined for the individual nodes. Therefore, executing equation (3) for every node contained in the neural network results in simulating the entire neural network for Δt. This process is referred to as a simulation step. In the simulation step, the following three operations must be performed for every node of the neural network.

(Vote phase)

The value (vote value) obtained by multiplying the result of applying the output function to the activity degree of a node by the link weight is conveyed to other nodes.

(Accumulation phase)

The vote values received from other nodes are accumulated.

(Update phase)

The activity degree of the node is updated according to the results of vote value accumulation.

Generally, neural network models, including the one proposed here, have the following attributes:

(1) Composed of many nodes (small calculation elements) and links (with weight) connecting the nodes.

(2) Each node changes its internal status according to the accumulation of numeric messages received from other nodes.

(3) Each node calculates its output value by applying an output function to its internal status. The output value is multiplied by the link weight, and the product is output as a numeric message to other nodes.

Neural networks with the above attributes can be simulated in the same manner as that proposed by us. Our simulator is aimed at simulating, at high speed, the neural network model described in this section. If its microprogram is modified, it can also be used to simulate other models.

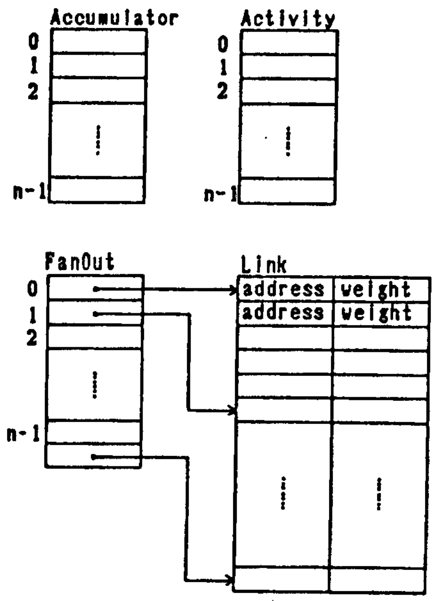


Figure 4. Simulation Data Control Format

4. Acceleration Measures and System Structure

The way in which the parallelism characteristic of the target neural network is extracted affects the performance of the simulator. To obtain a higher simulator speed, we have directed our attention toward the parallelism characteristic of the neural network and that of the simulation algorithm.

The nodes in a neural network can operate in parallel. Assigning a physical processor to each of the nodes will best enhance their parallelism characteristic. However, it is unrealistic, from both technology and cost standpoints, to assign to each node a hardware device capable of digitally performing the three types of operations for node simulation. We have adopted an approach by which the target network is divided into partial neural networks and more than one processor element (PE) is assigned to them. In this arrangement, although the nodes are processed successively within each processor element, the partial networks assigned to different processor elements are processed in parallel. This method of parallel processing has been put to practical use as a circuit division technique for logic simulation in the CAD field.

Of the three operations to be performed in one simulation step, the vote and accumulation phases can be executed in parallel. Each processor element has two processing units. They are used exclusively to process, in parallel, the above-mentioned two phases in the pipelining mode in order to extract the parallelism characteristic of the simulation algorithm.

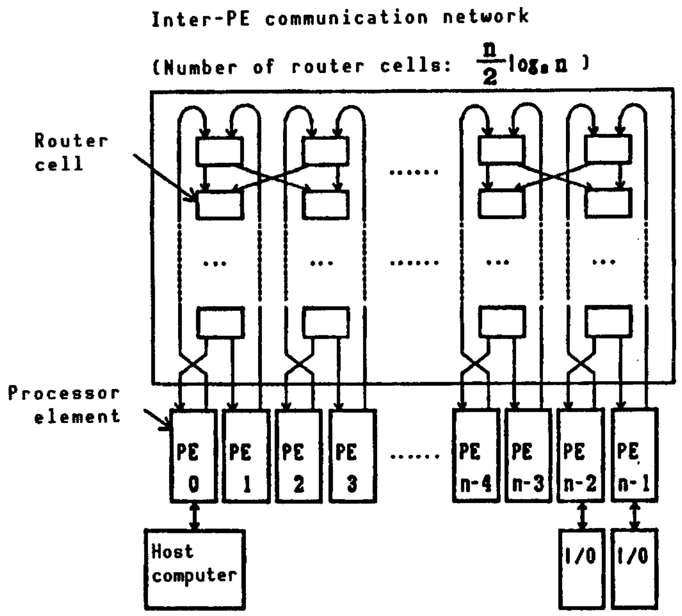


Figure 5. NeuMan System Configuration

The system structure of the NeuMan is shown in Figure 5, and a block diagram of the processor element is shown in Figure 6. The NeuMan consists of special-purpose processor elements (PEs) and an inter-PE communication network used for message transfers between processor elements. The neural network is divided into partial networks, and the partial networks are assigned to different processor elements. Each processor element to which a partial network has been assigned executes the vote, accumulation, and update phases for the nodes contained in the partial network in synchronism

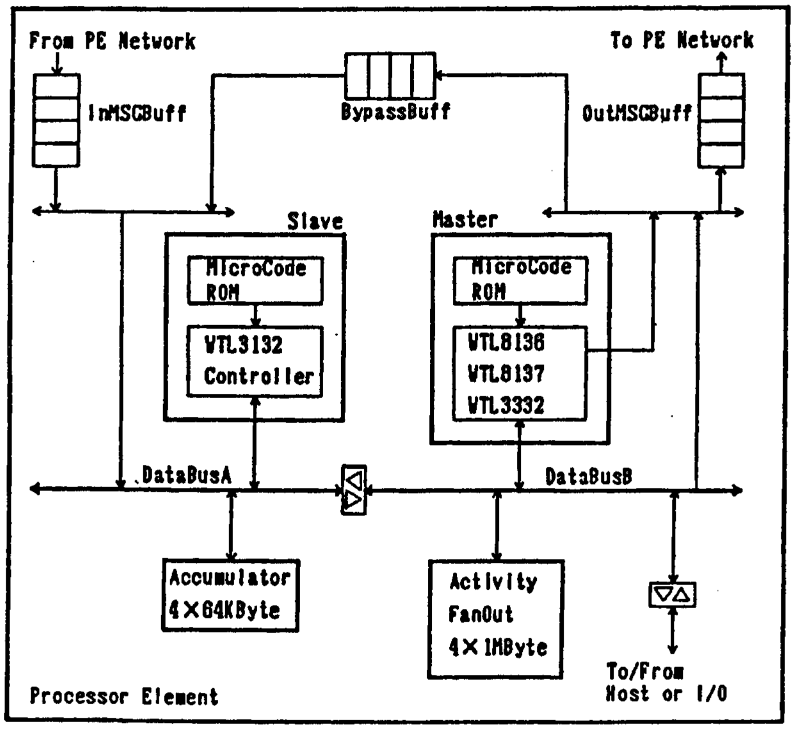


Figure 6. Processor Element (PE)

with other processor elements. If a node worked on by a processor element is linked with another node being worked on by another processor element, the former outputs a vote message to the inter-PE communications network. The vote message comprises the vote value of the node being worked on by the former processor element and the address of the node being worked on by the latter processor element. The inter-PE communications network transfers the message to the processor element corresponding to the node address included in the message. the processor element given the vote message locates the addressee node and adds the vote value to the accumulated vote value for the addressee node. This is who the vote and accumulation phases are executed. Each processor element has two processing units able to execute the above two phases in parallel in the pipelining mode. After executing the vote and accumulation phases, each processor element executes the update phase independently of other processor elements.

The inter-PE communications network is a multistage link network comprising router cells (2-x-2 switches). An inter-PE communications network to which as many as n processor elements are to be connected requires router cells equal in number to (n/2) multiplied by $\log_{2n}$.

The NeuMan serves as a back-end processor for the host computer. The host computer loads data for use in simulating a neural network into the processor elements via the inter-PE communications network. The host computer also controls the simulation, monitors the activity degree of each node during the simulation, and provides an interface between processor elements and the environment. Since each processor element has an I/O port, it can also be directly interfaced to the environment without requiring the service of the host computer.

5. Parallel Algorithm

In this section, the simulation of a neural network using a simulator structured as described in Section 4 will be explained.

Each processor element stores information on the partial network assigned to it in the activity (x) table, accumulator (x) table and fan-out table (a table listing the address of node y influenced by node x and the weight Wxy of the link between node x and node y), as shown in Figure 4.

Each processor element carries out simulation according to the above data structure to maintain synchronism with other processor elements. Step-based synchronism between processor elements can be achieved by inputting a special message (SYNC) for processor element synchronization to the inter-PE communications network without requiring any major synchronization operation to be performed.

The contents of processing performed by each processor element during each of the three phases are as follows:

(Vote phase)

```
FOR x IN  All nodes assigned to PEi  DO
      mx: = $\mu$ (Activity[x]);
      FOR link IN  All fan-out links of node x  DO
        MSGout.addr := link.address;
        MSGout.vote := link.weight X mx;
        put(MSGout);
put(SYNC);
```

(Accumulation phase)

```
MSGin := get();
WHILE MSGin $\neq$ SYNC DO
      Accumulator[MSGin.addr] :=
        Accumulator[MSGin.addr] + MSGin.vote;
      MSGin := get();
```

(Update phase)

```
FOR x IN  All nodes assigned to PEi  DO
      Activity[x] := Activity[x] +
        $\Delta$tx {Accumulator[x] - Activity[x]}/$\tau$;
      Accumulator[x] := 0.0;
```

(Description of symbols used above)

PEi:	Processor element i
Activity:	Activity degree of node
Accumulator[x]:	Accumulator of node
MSG.addr:	Address part of message

MSG.vote:	Vote part of message
link.address:	Node address of link
link.weight:	Weight of link
SYNC:	Synchronization message
get():	Message input
put(MSG):	Message output
Δt:	Unit of integration

In the vote phase, each processor element performs voting according to the activity table and the fan-out table for each node assigned to it. The output of a node is calculated by applying the output function μ to the activity degree of the node. The calculated output is multiplied by the weight of the link. The product (vote value) is combined with the address of the node receiving the vote, and they are output as a vote message to the inter-PE network. When the processor element completes the voting procedures for all the nodes assigned to it, it outputs a synchronization message SYNC to the inter-PE network.

In the accumulation phase, the processor element accumulates the values specified in the vote sections of the vote messages received via the inter-PE communications network in the vote accumulators for the corresponding nodes specified in the address sections of the vote messages received. This accumulation operation is continued until the processor element receives a SYNC signal.

Upon termination of the vote and accumulation phases, the processor element begins to execute the update phase. During the update phase, it updates the activity degrees of all the nodes assigned to it while referring to the activity table and the accumulator table.

6. Processor Elements

A block diagram of the processor element is shown in Figure 6. Each processor element has two processing units, i.e., master and slave processing units, used to execute the vote phase and the accumulation phase in parallel. Both the master and the slave processing units are controlled by microprograms. Therefore, if the microcodes used in the microprograms are changed properly, the processor element can be used to simulate different neural network models or to conduct different kinds of research. The master processing unit consists of a 32-bit floating-point processor, an integer processor and a sequencer. It can access all resources in the processor element. It is even possible to carry out simulation using only this master processing unit. The slave processing unit is used to execute the accumulation phase in parallel with the execution of the vote phase carried out by the master processing unit. It consists of a 32-bit floating-point processor and a simple controller. Executing the vote phase and the accumulation phase--the two phases occupying a substantial portion of the neural network simulation process--in parallel, using the master and slave processing units in the pipelining mode, results in faster simulation. After the vote phase and the accumulation phase are completed, the update phase is executed by the master processing unit.

The activity degree of each node and the corresponding accumulated vote value, obtained by adding up the vote values received from other nodes, are entered in the node's activity table and accumulator table, respectively, as 32-bit floating-point values. Information on the links between nodes is entered in the fan-out table. The information comprises node addresses (each node address consists of an 8-bit processor element identification number and a 16-bit intraprocessor element address) and link weights (32-bit floating-point values). Each processing unit can independently access the activity table and the fan-out table during the vote phase and the accumulator table during the accumulation phase. During the update phase, both the accumulator table and the activity table must be referred to. During this phase, the master processing unit can access the two tables while the slave processing unit is inactive.

Each processor element has an I/O port connected to the inter-PE network and uses it to exchange vote messages with other processor elements. The two processing units incorporated in each processor element are connected by a vote message bypass. The bypass is used to transfer vote messages on nodes within the same processor element. The use of the bypass results in reducing the traffic through the inter-PE network.

The activity table and the fan-out table have a combined capacity of 4 M bytes; the accumulator table has a capacity of 256 K bytes. They enable the corresponding processor element to simulate a neural network comprising up to 64 K nodes and 440 K links. Each processor element can achieve a processing speed of about 1 M links/s using its two processing units to carry out parallel and pipeline processing.

Each of the master and slave processing units uses a 32-bit floating-point operation unit that operates based on a 10-MHz (100 ns) clock. The operation unit can perform multiplication and addition against one instruction and is capable of pipeline processing at 100-ns intervals. The memory used is a DRAM with an access time of 100 ns and a cycle time of 220 ns. Each processor element is capable of simulating a neural network comprising up to 64 K nodes and 440 K links. When a processor element simulates a neural network of the maximum size mentioned above, it takes the following numbers of clocks to execute the three phases comprising one simulation step:

• Vote phase:	3,648 K clocks
• Accumulation phase:	1,920 K clocks
• Update phase:	384 K clocks

Since the vote phase and the accumulation phase are executed concurrently, the total number of clocks required to execute one simulation step becomes:

$$\max[\text{vote phase, accumulation phase}] + \text{update phase} = 4{,}032 \text{ K clocks}$$

Since one clock equals 100 ns and 1 K equals 1,024, the time required to execute one simulation step is 413 ms. Therefore, the processing speed is 1 M (440 K/413 m) links/s.

7. Inter-PE Communications Network

During neural network simulation, such operations as data transfer, simulation control by the host computer, processor element synchronization and loading neural network data into processor elements are performed on a message basis using the inter-PE communications network.

The message format is shown in Figure 7. The messages have a variable word length (a word consists of 16 bits). Each message consists of a header (1 word) and main parts (0-127 words). The header indicates the message type, the message receiver address and the message length.

Header	Main part 1	Main part 2	• • •	Main part n

(n = 0-127)

Figure 7. Message Format

During a neural network simulation, vote values are transferred between nodes using vote messages. The vote message format is shown in Figure 8. A vote message must contain the information identifying the node that is to receive the message and a vote value. The information identifying a node in a neural network must contain the address of the processor element to which the node has been assigned and the address in the processor element of the node.

1st word		2d word	3d word	
Header		Node address	Vote value	
PE address	Message length		Charac-teristic	Mantissa part
(7)	(7)	(16)	(8)	(8)

Figure 8. Vote Message Format

In a vote message, the header contains the processor element address. Since a processor element takes care of up to 64 K nodes, the section used for node address indication in a vote message comprises 16 bits. The vote value is represented by a real number complying with the floating point format set by IEEE. The IEEE format consists of a total of 32 bits, i.e., an 8-bit characteristic and a 24-bit mantissa part. In neural network simulation, the value precision is not very important. A neural network simulation experimentally conducted by running a simulator on a LISP machine whose mantissa for the vote messages had been reduced to 8 bits did not produce any change in results. Therefore, of the 24-bit mantissa included in the vote value section of each vote message, only the 8 high-order bits are used, even though the processor element performs on a 32-bit basis. The vote value section of the vote message consists of 16 bits, i.e., 8 bits each for an exponent and a mantissa. With the mantissa reduced from 24 to 8 bits, the vote message section has been expanded to 3 words, resulting in higher message transfer speed. It is also possible to use 32-bit precision vote messages to carry out simulation with higher precision.

The inter-PE communications network is a multistage link network comprised of router cells. A router cell block diagram is shown in Figure 9. Each router cell is a switch with two input ports, A and B, and two output ports, X and Y. It contains four message buffers. Each port is 16 bits wide. The messages received are entered in the buffers selected according to the specified addresses. The buffer selection is controlled at the output ports of the router cell during the preceding stage. A message coming through input port A to be sent to output port Y, for example, is entered in the buffer AY.

Each output port is provided with a multiplexer. The messages coming through port A or B are advanced to an output port.

When a message exits an output port, it is input to an input buffer of the next-stage router cell. During this process, the input buffer selection is made by the output port.

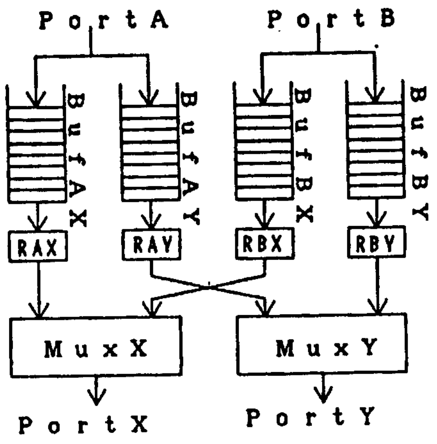


Figure 9. Router Cell

The router cell shown in Figure 9 can continue operation unless the two buffers inputting messages to the two multiplexers are both empty. (Here, it is assumed that the input buffers, to which the output ports of this router cell are connected, of the next-stage router cell have not overflowed. This assumption is proper as explained later.)

To enable a processor element to operate at a rate of 1 M links/s in simulating a neural network, the router cells must handle three-word vote messages to be exchanged between nodes at a rate of 2 M links/s. In other words, each router cell must receive messages, on the average, every microsecond, and output the messages received to the two output ports at the same message density. The router cells are designed to be capable of operating continuously without causing any buffer to overflow unless the message density exceeds 1.5 messages/μs at each input port.

In the following, the message processing capacity of the router cell will be analyzed. Assume that the messages received through each input port constitute Poisson-type arrivals at an average rate of λ messages/s), and that each multiplexer performs an exponent-type service at an average rate of μ (messages/s). Although, in reality, the processing time of a multiplexer is almost constant ($1/\mu$), its service is assumed to be of an exponent type here to avoid making the analysis too complicated. Analysis based on this assumption produces more exact results than does that based on the assumption that the multiplexer service is constant. Therefore, adopting the above assumption regarding the multiplexer service guarantees being on the safe side when conducting analysis.

When a message is input through an input port, it is entered on one of the two buffers according to the destination address specified. Therefore, the messages arriving at a buffer constitute Poisson $\lambda/2$ (messages/s). Since each multiplexer handles the messages output from two buffers, the message processing capacity of a router cell can be analyzed through analyzing the system shown in Figure 10.

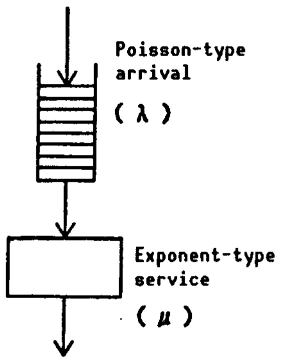


Figure 10. Poisson-Type Arrival/Exponent-Type Service System

The system shown in Figure 10 is a typical queue system. The system analysis results will be explained. When it is assumed that the traffic density $\rho = \lambda/\mu$, the probability of the number of messages being n in the system in a steady state is expressed as follows:

$$p_n = (1 - \rho)\ \rho^n$$

If the number of messages exceeds n while the buffer length is n, not all the messages can be entered in the buffer. The probability $p_{1>n}$ of the buffer length exceeding n can be calculated as follows:

$$\begin{aligned} p_{1>n} &= p_n + p_{n+1} + \cdots \\ &= (1 - \rho)\ \rho^{(n+1)} \sum_{1=0}^{\infty} \rho^1 \end{aligned}$$

If

$$\sum_{1=0}^{\infty} \rho^1 = 1/(1-\rho)$$

is substituted into the above equation, the following result is obtained:

$$P_{1>n} = \rho^{(n+1)}$$

At the design stage, it is specified that $\lambda = 1$ M (messages/s) and that $\mu = 1/0.6$ M (messages/s). From this $\rho = 0.6$ and $n = 512/3$ (a buffer consists of 512 words of 16 bits each and a simulation message comprises 3 words) = 170. The probability of the input buffer overflowing becomes:

$$P_{1>170} = \rho^{171} = 0.6^{171} = 1.9 \times 10^{-38}$$

Probability of this magnitude is negligible.

If λ is 1.5 M (messages/s) and ρ is 0.9, the probability becomes:

$$P_{1>170} = \rho^{171} = 0.9^{171} = 1.5 \times 10^{-8}$$

Probability of this magnitude is also negligible.

Therefore, the router cell can perform message switching without causing either buffer to overflow, even if it receives messages through both input ports at a rate of 1.5 M (messages/s) each.

The processor elements are synchronized during every step by the use of messages. The synchronization is achieved in the following way without involving any special large-scale processing.

At the end of the vote phase of a simulation step, each processor element outputs a synchronization message SYNC to the network, signaling to the router cell that it has no more messages to be sent out in the current step

The SYNC message reaching the input port of the router cell is input to the two input buffers connected to the input port. The messages subsequently arriving at the input port should be vote messages issued during the following step. When a multiplexer receives a SYNC message from one of the two input buffers, it knows that no messages to be transferred during the current simulation step are left in the input buffer, and continues processing the messages coming from the other input buffer. When it subsequently receives a SYNC message from the other input buffer, it outputs a SYNC message from its output port to let the next-stage processor element know that no messages remain for transferral during the current simulation step.

When a processor element receives a SYNC message from the inter-PE network, it knows that no more messages will be forwarded to it during the current simulation step, and it terminates the accumulation phase. After terminating the vote phase and the accumulation phase, it executes the update phase. Upon completion of the update phase, it terminates the current simulation step and then starts the next simulation step.

8. Conclusion

We have proposed a special simulation machine capable of simulating a large-scale neural network at high speed. It is designed to perform parallel processing using more than one processor element and utilizing the parallelism characteristic of the neural network simulation algorithm, as well as that of the neural network.

We intend to fabricate a simulator system incorporating four to eight processor elements and to test it to verify the propriety of the architecture. We also plan to develop a neural network programming environment tightly coupled with the NeuMan on the host computer. The neural network programming environment must contain a neural network description language, a compiler, a neural network split loader, and a monitor. Among them, the neural network description language, that must be highly descriptive, and the neural network split loader assume particular importance.

To enable the NeuMan to simulate a neural network with high efficiency, it is necessary to split the neural network into partial networks, minimizing the number of messages to be exchanged between processor elements. When the neural network program is written in the neural network description language, it is divided into function modules. The information used for modularizing the program can be utilized in dividing the neural network.

References

1. Kajiwara and Tsuji, "Neural Network Inference System," material of the knowledge engineering and artificial intelligence study section of the Information Processing Society of Japan, 45-10 (1986).

2. Kajiwara, Nakada, and Koike, "Neural Network Simulator in Special Parallel-Processing Machine MAN-YO," Text prepared for 35th national meeting of the Information Processing Society of Japan, (1987).

20109/9365

Optically Connected 3-D VLSI Sorting Scheme

43067589b Tokyo KONPYUTA AKITEKUCHA SHINPOJIUMU RONBUNSHU in Japanese
17-18 May 88 pp 55-64

[Article by Makoto Hasegawa, College of Engineering, Shizuoka University, and Susumu Horiguchi and Yoshiharu Shigei, Faculty of Engineering, Tohoku University: "Sorting Scheme in Optically-Connected Three-Dimensional VLSI Architecture"]

[Text] **Abstract**

Data totaling N in number can be sorted in time, $T = O(\log^2 N)$, $AT^2 = O(N \log^5 N)$, $VT^{3/2} = O(N \log^4 N)$, through the realization of bitonic sorting in an optically-connected three-dimensional VLSI architecture. This performance is superior to the theoretical limit value, Ω $(N^2 \log N)$, of the area-time complexity (AT^2) determined by Thompson for sorting in a two-dimensional VLSI architecture.

The superior sorting performance indicated above is attributable to the utilization of the freedom offered by the new dimension introduced in the three-dimensional architecture.

It has become clear that the adoption of the three-dimensional VLSI architecture may result in the realization of VLSI performance far superior, with regard to some important problems, to that of two-dimensional VLSIs. This potential of the three-dimensional VLSI architecture appears to suggest that the efforts made to develop three-dimensional VLSIs will be adequately rewarded.

1. Introduction

The greatness of the potential of the three-dimensional VLSI is being increasingly recognized. This status is described well in the commentary (KURO86) by Kurokawa, et al. A three-dimensional architecture allows more flexible problem mapping than does a two-dimensional one. In this connection, it has been known (HASE86) that the computational complexity of FFT [fast Fourier transform] can be greatly reduced by adopting a properly arranged three-dimensional system architecture. In this paper, the computational complexity involved in sorting on a three-dimensional VLSI will be studied, and it will be explained that a fine sorting performance can be achieved using a three-dimensional VLSI architecture. The introduction of VLSIs capable of sorting a large quantity of data at high speeds is strongly desired. In the present situation involving the two-dimensional VLSIs, however, it is difficult to realize a satisfactory sorting performance using a realistic portion of the chip surface.

The introduction of VLSIs has made it possible to form digital systems on chips, leading to substantial improvements in the operating speed of semiconductor devices. However, inside the VLSI chips, the operating speed is restrained due to the drive delay caused when long-distance wiring used for wide-area communications are charged and discharged. In fact, a significant portion of the VLSI chip surface is used for wiring (MURO82).

Such being the case, a competitive race toward the development of an algorithm that would enable the above problem to be solved on a local communications basis is under way. With regard to some problems, remarkable achievement has already been made. However, as research in this area progresses, it has come to be recognized that problems that can be solved on a local communication basis are the exception and, moreover, that many problems that exist whose efficient solutions appear to be fundamentally dependent on wide-area communications. Furthermore, it has been found that many important operations for which acceleration is urgently required present problems such as those stated above. Among these operations, FFT and sorting are typical.

To cope with the situation, it is necessary to devise a measure to improve the efficiency of wide-area communications. A three-dimensional VLSI architecture is a prime candidate for adoption as a means of realizing the desired efficiency improvement.

In the following part of this article, it will be explained that data totaling N in number can be sorted in time, $T = O(\log^2 N)$, $AT^2 = O(N \log^5 N)$, and $VT^{3/2} = O(N \log^4 N)$, through the realization of bitonic sorting in an optically connected three-dimensional VLSI architecture. These results are better than the theoretical limit value, $\Omega\ (N^2 \log N)$, of the area-time complexity (AT^2) determined by Thompson for a two-dimensional VLSI system.

It also signifies that, when based on the performance of a single processor, the system performance may be enhanced by more than N times when using N times as many resources. This possibility is due to the additional dimension introduced as a result of the adoption of a three-dimensional architecture. The freedom in the new dimension can be advantageously utilized (needless to say, this does not refer to the superlinearity within the same system). As far as we know, the FFT (HASE86) and sorting (HASE86B) operations performed on three-dimensional VLSI systems are the first examples in which the above-stated possibility has been materialized.

2. Bitonic Sorting

The sorting algorithm adopted for the present system is bitonic sorting. This sorting method was originally devised by Batcher (BATC68). However, the present discussion will be advanced based on the bitonic sorting studies made by Stone (STON71) and Knuth (KNUT73). In this article, a bitonic sequence refers to a composite sequence formed by joining two sequences of the same length--one of ascending order and the other of descending order. When a sequence $\{a_1, a_2, \ldots, a_{n-1}, a_n, \ldots, a_{2n}\}$ is a bitonic sequence satisfying the relationship of $a_1 \leq a_2 \leq \ldots \leq a_n$, $a_{n+1} \geq a_{n+2} \geq \cdots \geq a_{2n}$, the two sequences generated by applying the operations $MIN_i = \min(a_i, a_{n+i})$ and $MAX_i = \max(a_i, a_{n+i})$ to the sequence realize the following relationships:

(I) The new sequences generated, $\{MIN_1, MIN_2, \ldots, MIN_n\}$ and $\{MAX_1, MAX_2, \ldots, MAX_n\}$, are both bitonic.

(II) Any element of the sequence $\{MAX_1, MAX_2, \ldots, MAX_n\}$ is greater than any element of the sequence $\{MIN_1, MIN_2, \ldots, MIN_n\}$.

Therefore, the original bitonic sequence can be sorted by recursively applying the above operations, O(log N) times, to the partial bitonic sequence equal to one-half of the original bitonic sequence. With regard to the unit of operation in this case, it is enough if, for the target bitonic sequence with a length of m, compare-exchange operations can be performed between the sequence elements arranged m/2 apart (Figure 1).

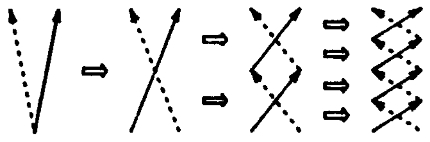


Figure 1. Bitonic Sorting Procedure

Figure 2 shows the flow of signals in bitonic sorting performed for eight inputs. The squares in Figure 2 represent compare-exchange units. The two inputs shown on the left-hand side of each compare-exchange unit are output to the two output terminals on the right-hand side of the unit after being rearranged in the direction of the arrow shown in the square according to a comparison of their values performed in the compare-exchange unit.

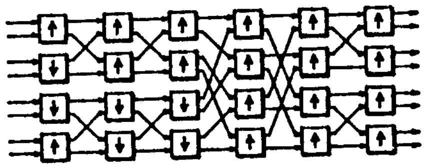


Figure 2. Signal Flow in Bitonic Sorting

3. System Structure

First, efficient sorting is dependent on wide-area communications. As long as it is based on a two-dimensional VLSI architecture, it is subjected to the drive delay resulting from the use of long-distance wiring.

Now, a system structure that can realize a sorter for N inputs will be discussed based on the signal flow shown in Figure 2. The system comprises the processing stages for executing compare-exchange operations. The processing stages total O(log N) in number and are sequentially arranged. They are optically connected through identical path-length networks (Figure 3). Each processing stage is equivalent in function to a processing row shown in Figure 2. It includes independent compare-exchange elements, totaling N in number, that operate in parallel. The inputs are given from the left-hand side, as viewed based on Figure 3, and the final output is obtained on the right-hand side.

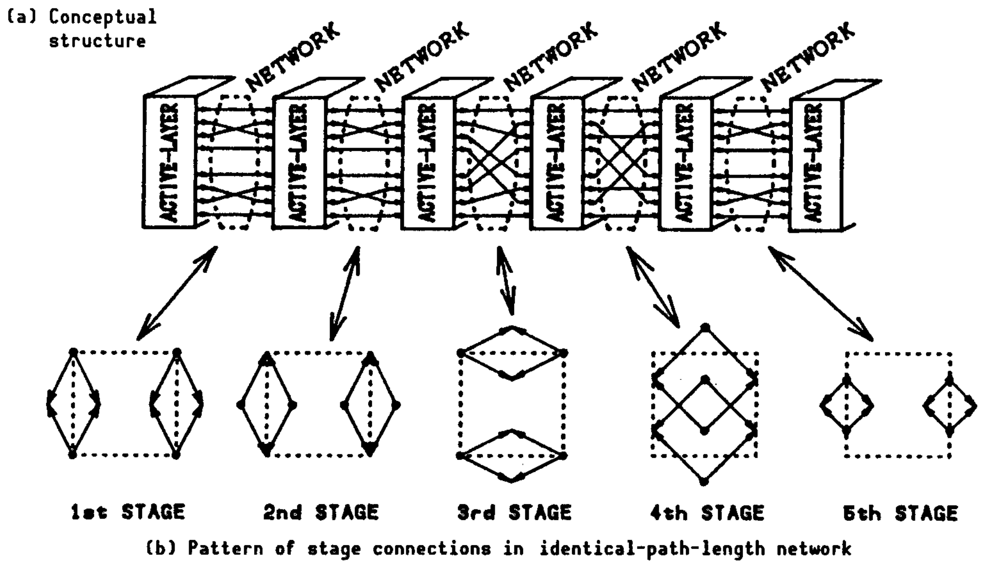


Figure 3. Sorting System Structure on Three-Dimensional VLSI Architecture

Identical-path-length network

The processing stage connection network adopted for this system is an identical-path-length network (HASE86b). In the identical-path-length network, the path length between any two points connected during the same processing stage is constant. In this network, optical signals are used as communication media in the connection stage. The path connecting any two nodes is not blocked by any other path connecting other nodes. Therefore, in this network, such optical elements as optical waveguides, free space, and transparent flat plates may be used arbitrarily.

Since the signals propagated along different paths in this network are free from interference, two or more data streams can exist in parallel in the same network if it is physically feasible to produce the data streams. In addition, in this network, it is permitted to transmit one bit of data along a path as soon as the transmission of the preceding bit of data is completed on the same path. Therefore, two or more bits of data may exist concurrently, arranged in the order of transmission, along the same path. Each node in each processing stage can compare, exchange and transmit, i.e., it compares the two signals received from two nodes of the preceding processing stage via the connection stage, orders them according to the comparison results and transmits them to the corresponding nodes of the next stage. Each processing stage contains compare-exchange nodes totaling N in number. All the nodes operate in parallel.

Processing Stage Structure

Each compare-exchange element included in each processing stage is composed of a photo detector layer, a compare-exchange layer, and a light emitter layer (Figure 4). In the light emitter layer, the optical signals received from the preceding stage are converted into electric signals and are sent to the compare-exchange layer. The results of the signal comparison conducted in the compare-exchange layer are converted into optical signals in the light emitter layer and are output to the next stage. Each processing stage constitutes an array comprised of compare-exchange elements totaling N in number and is capable of parallel operation. Physically, this array can be formed as a three-dimensional VLSI consisting of three layers, i.e., a photo detector layer, a processing layer, and a light emitter layer. We have decided to study the structure in which each operation stage has an array of compare-exchange elements like the one described above and connection stages are stacked alternately.

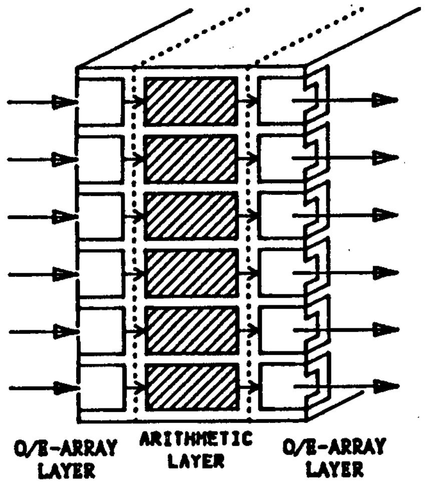


Figure 4. Processing Stage Structure

4. Evaluation Criteria

In this section, the volume performance advocated by Rosenberg (ROSE83) will be discussed, in addition to the time computation complexity and the area-time performance. The volume performance is a concept expressing the cost of the operation performed, using a uniform material, according to the amount of the material used. It can be said of the evaluation criteria in general use today that more discussion will be required to determine a definition of the time performance in order to express the processing capacity of a system and the criteria for assessing the problems related to the new type of three-dimensional architecture, which is somewhat different from the conventional three-dimensional IC architecture.

As a yardstick of the difficulty encountered in realizing an algorithm, the area evaluation appears more appropriate than the volume performance. As an index of the final product cost, the volume evaluation--a material cost indicator--will be effective when an adequate yield can be secured (HIBI86).

It will be explained later that the three-dimensional VLSI architecture has proven superior to the two-dimensional VLSI architecture, no matter which evaluation criterion in use today is used in comparing the two.

5. Hypotheses for Optically Connected VLSI Models

The hypotheses set for conducting the time and area performance evaluation of optically connected VLSI models will be explained.

> Hypothesis 01: Data transfer rate (optical connection system)
>
> The transfer of 1 bit of data along an optical propagation path requires a unit length of time.

The above hypothesis holds if an adequate length of data is to be bit-serially transferred using a pipelined drive circuit for the electrooptic converter.

> Hypothesis 02: Drive delay in optical connection
>
> The attenuation along the optical propagation paths running where circuit elements are optically connected is adequately low and the distance-dependent drive delay is negligible.

This hypothesis can be true when low-loss optical waveguides or free space can be used as optical communications channels.

If the rate of attenuation along the optical propagation paths is not adequately low, then it is necessary to take into account the drive delay $O(L)$ for propagation distance L (HASE87).

> Hypothesis 03: Propagation delay in optical connection
>
> The propagation delay to the limitation of the light velocity is negligible.

In wire-connected VLSI models, the time required for information transfer itself (propagation delay) has been regarded as negligible (it may be more appropriate to say that the propagation delay has not even been taken into consideration). Since this analysis is aimed at comparing different systems, the propagation delay is also regarded as negligible for optically-connected three-dimensional VLSIs.

6. Results Obtained for Wire-Connected Two-Dimensional VLSIs

For the purpose of comparison with the optically-connected three-dimensional VLSI architecture, the computational complexity, determined by Thompson, of sorting in a wire-connected two-dimensional VLSI architecture will be shown first. Then, the extent to which it is affected by the current density limitations pointed out by Card will be discussed briefly.

6.1 Results Obtained by Thompson

(Theoretical lower limit value of area-time complexity (AT^2))

Thompson previously reported (THOM80) that, with regard to the ($area*time^2$) performance in the sorting of data totaling N in number, Ω ($N^2 \log^2 N$) was the best solution available. According to his latest research results (THOM83), the theoretical lower limit value is Ω ($N^2 \log N$).

(Sorting in shuffle-connected system)

Bitonic sorting in a shuffle-connected system can be realized at $T = 0(\log^3 N)$, $A = 0(n^2/\log^2 N)$, $AT^2 = 0(N^2 \log^4 N)$. (THOM83)

(Sorting in mesh-connected system)

Sorting in a mesh-connected system can be realized at $T = 0(N^{1/2})$, $A = 0(N \log^2 N)$, $AT^2 = 0(N^2 \log^2 N)$. (THOM83)

6.2 Card's Theorem (CARD86) and Its Effect

The theoretical lower limit value determined by Thompson is based on Mead's report (Mead82) (Mead80) that the delay time involved in driving a wiring with length L could be reduced to 0(log L) using an exponent-type driver. However, Card pointed out (Card86) that, since the current density along the wiring on contemporary VLSIs is already close to the threshold value causing electromigration, the adoptability of an exponent-type driver designed to enable as large a current as desired to flow through the wiring was problematical. When this is taken into consideration the drive delay in a wiring with length L should be as follows:

> (Theorem 1) In an arbitrary VLSI layout, the delay caused in charging and discharging with length L is 0(L). (CARD86)

The ways in which two representative methods of sorting on VLSIs will be affected by the restrictive condition given by the above theorem will be discussed next.

(Sorting in mesh-connected system at limit current density)

Bitonic sorting can be performed at $0(\log^3 N)$ when the current density limit does not need to be considered in performing the compare-exchange steps. Between compare-exchange steps, a $0(N^{1/2})$ step is required in performing the data transfer in preparation for the next operation. The overall delay time involved in sorting is $O(N^{1/2})$.

However, when it is necessary to consider the current density limit, a problem arises in connection with the driving of wiring between processors. Since the wiring length is $O(\log N)$, the routing time per stage should be estimated at $O(\log N)$. The area-time complexity under these conditions is as follows:

Sorting in mesh-connected system at limit current density:

$A = O(N \log^2 N)$, $T = O(N^{1/2} \log N)$, $AT = O(N^2 \log^4 N)$.

(Sorting in shuffle-connected system at limit current density)

In a shuffle exchange connected system, the maximum wiring length between operation stages is $O(N/\log N)$. Therefore, the routing delay in the connection network is $O(N \log N)$. Therefore, the complexity becomes:

Sorting in shuffle-connected system at limit current density:

$A = O(N^2 \log^2 N)$, $T = O(N \log N)$, $AT^2 = O(N^4)$.

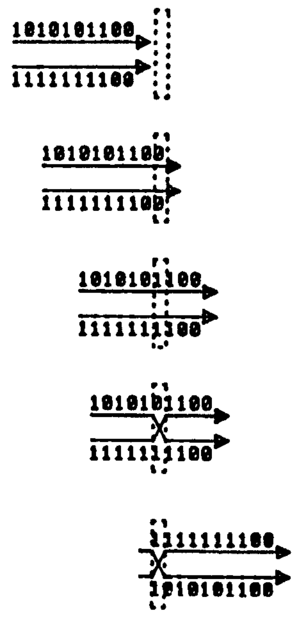
When the current density limit must be taken into consideration, sorting in a mesh-connected system can be executed at higher speed than can that in a shuffle-connected system.

7. Performance Evaluation

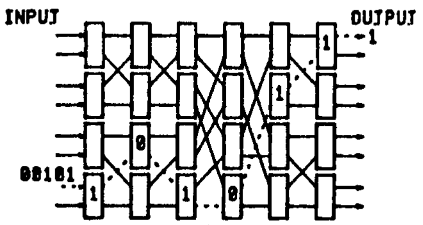
(Pipeline-type bit-serial compare-exchange unit)

A bit-serial compare-exchange unit capable of pipeline operation for the execution of bitonic sorting in the three-dimensional system discussed here can be realized as follows: number of gates = $O(1)$, area $A = O(1)$ and delay $T = O(1)$. This unit constitutes a module with bit-serial input terminals corresponding to two input streams, a and b, and two output terminals corresponding to the two outputs--max(a,b) and min(a,b). The data formats of the input and output data streams are represented by serial bit strings. The heading bit of each of the serial bit strings is called the most significant bit (MSB). The comparison, exchange, and transmission operations to be performed to execute bitonic sorting will be discussed next. The data words must be input bit-serially, with the highest-order bit leading the way. Each compare-exchange unit must be capable of comparing two input values and changing their output direction according to the results of the comparison within the unit length of time per bit. During the time between when the input of data from two input data streams to a compare-exchange unit is started and when the first unmatched pair of bits is received by the compare-exchange unit, the compare-exchange unit copies the bits input from both streams and outputs the copied bits. When the first unmatched pair of bits is received, the compare-exchange unit compares them to determine which bit is larger, and changes the output direction according to the results of the comparison. Once it changes their output direction, the compare-exchange unit does not change the output direction again until the data words are finished. Until the data output direction is changed for the pair of input streams, the values output from the compare-exchange unit are always identical for the two output terminals (even though the data output from the two output terminals are from different sources). Therefore, as far as the data values are concerned, the compare-exchange unit can perform the data comparison and exchange in the manner described above.

If the above-described operation is performed during every operation cycle for every bit of data, the minimum cycle time is the time required to compare two bits of data and to change their output direction. When this method is used, it is possible to generate 1 bit of output data every unit of time.



(a) Compare-exchange unit



(b) Data passage

Figure 5. Pipelining Using Bit-Serial Compare-Exchange Units

If the possibility exists that the routing time will vary with the route in the connection network, it is necessary to enter each pair of bit streams received from the preceding stage into buffers and then to input them to the corresponding compare-exchange unit, in order, beginning with the leading bits, while maintaining synchronism between the two input data streams. The necessity of providing for absorbing the skew in this case limits the sorting performance.

Unlike the above system, the sorting system we have devised incorporates an identical-path-length routing network so that no problems develop from the signal skew between input data streams. In this system, the two input data streams may be regarded as being kept in perfect synchronism. In this sense, the identical-path-length routing network assumes fundamental importance to this sorting system.

Routing Time Requirement

From the foregoing hypotheses, the time required for routing in an identical-path-length network can be established at $O(1)$. In bit-serial data transfer, as hypothesized, a data word comprising $O(\log N)$ bits takes time $O(\log N)$ to pass through the network. However, since the propagation delay in the network is $O(1)$, and both the compare-exchange units and the routing network participate in pipeline operation in the present sorting system, the time required for routing does not exceed the time taken by the lead signal in moving from the input end to the output end of the network, i.e. $O(1)$.

From the above, in each processing stage, one operation unit can be executed using bit-serial compare-exchange units totaling N in number (area: $O(1)$) and taking time $O(1)$. The time taken to transfer the data input to the connection network to the next stage is $O(1)$ if the light velocity is high enough to make the propagation delay negligible. When this process is repeated for stages totaling $O(\log^2 N)$, the total area requirement is $O(N \log^2 N)$, the total time requirement is $O(\log^2 N)$ and the output period is $O(\log N)$. When the present sorting system is evaluated by Thompson's criteria (THOM83), the time computational complexity is $T_d = O(\log^5 N)$ and the area computational complexity is $A_d = O(N \log N)$. The area-time computational complexity is $AT_d^2 = O(N \log^5 N)$. This value suggests the possibility for great performance improvement, in light of the fact that the area-time computational complexity determined by Thompson for two-dimensional VLSIs is $AT_d^2 = \Omega (N^2 \log N)$.

Many differences of opinion may be expressed when determining what quantities should be used when comparing two-dimensional and three-dimensional VLSIs. Rosenberg (ROSE83) has determined that the evaluation quantity for three-dimensional VLSIs, corresponding to AT^2 for two-dimensional VLSIs, is $VT^{3/2}$. If this theory is followed, $VT_d^{3/2} = O (N \log^4 N)$. Not that, since the present sorting system can handle instances totaling $O (\log N)$ simultaneously, the volume per instance, $V_d = O(N \log N)$, has been applied here.

Bitonic sorting in a mesh-connected network is a nearly ideal sorting method that can be used without occupying an unrealistically large area on a two-dimensional VLSI. Its area performance is $A = O(N \log^2 N)$, time performance is $T_d = O(N2)$ and area-time performance is $AT_d^2 = O(N^2\log^2N)$. The superiority of the present sorting method is obvious from these figures, too.

A summary of the above discussion appears in Table 1. The time performance of sorting on wire-connected two-dimensional VLSIs has been evaluated based on the assumption that the drive delay for a path length L is $T = O(L)$. Recent study results, however, indicate with increasing clarity (CARD86) that it is extremely difficult to make the cost of practical wide-area communications (drive delay) smaller than $T = O(L)$. As a result, it has become necessary to review the area-time evaluation so far referred to of sorting on a two-dimensional VLSI. This point is only partly reflected in Table 1. When this point is taken fully into account, the advantages of sorting on a three-dimensional VLSI over those of sorting on a two-dimensional VLSI will be better recognized.

Table 1. Area-Time Performance of Various Sorting Methods

Sorting method	Area efficiency (A_d)	Time efficiency (T_d)	AT_d^2	Source
Bitonic sorting on optically-connected three-dimensional VLSI				
(Volume evaluation)	V_d: $N \log N$	$\log^2N$	$VT^{3/2}$: $N \log^4N$	
(Area evaluation)	$N \log N$	$\log^2N$	$N \log^5N$	
Theoretical lower limit (for two-dimensional VLSI)	--	--	$\Omega\ (N^2\log N)$	(THOM81)
<When current density is limited>				
N-proc.bitonic.Mesh	$N \log^2N$	$N^{1/2}\log N$	$N^2\log^4N$	
N-proc.bitonic.S-E	$N^2/\log^2N$	$N \log N$	N^4	
<When current density need not be considered>				
N-proc.bitonic.Mesh	$N \log^2N$	$N^{1/2}$	$N^2\log^2N$	(THOM83)
N-proc.bitonic.S-E	$N^2/\log^2N$	$\log^3N$	$N^2\log^4N$	(THOM83)
Single processor	$\log N$	$N \log^2N$	$N^2\log^5N$	(THOM83)

8. Discussion of Realization of Proposed Sorting Scheme

Bitonic sorting in a three-dimensional VLSI architecture has been discussed without giving much consideration to its feasibility. In this section, how the sorting will be realized is discussed by taking into consideration the device techniques that are likely to become available in the near future (HASE86b). From the viewpoint of design elegance, a device structure in which compare-exchange processing stages and connection stages are alternately stacked in the order in which they are passed by signals seems attractive. Materialization of this device structure requires a technique that would enable the formation of a three-dimensional VLSI consisting of numerous active element layers. When two or more element layers are put together to form an integrated device, layers stacked later are likely to exhibit sharply higher defect ratios. With the device techniques presently available, it is extremely difficult to overcome this problem. Even if three-dimensional VLSIs, as described above, can be fabricated, they may pose problems with regard to heat radiation and production yields. Fortunately, the optical connection stage is required only as a medium, so it does not need to be an integral part of the three-dimensional VLSI from the beginning. This point constitutes a fundamental difference between the optically connected VLSI and the wire-connected VLSI. It is possible to fabricate processing stages with a compare-exchange function and an optical input-output function, with optical connection stages to be used separately as a communications medium, then test them and put together only those that have passed the performance test. Separate preparation of different parts with different functions will result in higher production yields and, eventually, in a much lower defect ratio during the final assembly stage. Chips may be compared in terms of the total active-region area. In a well-known method of rough evaluation, if chips, each with an area of A, have defects totaling, on the average, N per unit area, the probability P_0 at which defectless chips are found among them is given as

$$P_0(NA) = e^{-NA},$$

i.e., when the area exceeds a certain value, the yield starts dropping sharply. The system being discussed here will, when materialized, constitute a large-scale device with a marginal (if assessed based on the present technology level) integration density. Therefore, the above-described approach in which the processing stage and optical connection stage are prepared separately will produce an effect that cannot be ignored.

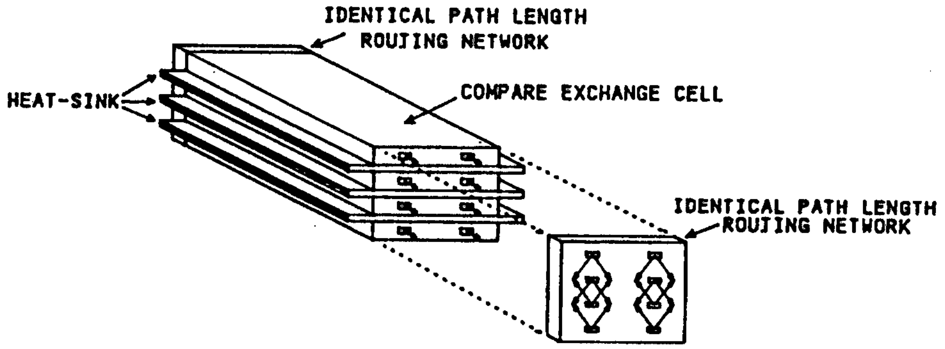

IDENTICAL PATH LENGTH ROUTING NETWORK
HEAT-SINK
COMPARE EXCHANGE CELL
IDENTICAL PATH LENGTH ROUTING NETWORK

Figure 6. Possible System Structure

Any actual attempt to fabricate this system will encounter many more problems. Since the system includes many elements that generate a large amount of heat, devising an efficient heat radiation measure may become a major hurdle to be cleared. A possible solution to the problem is to slice the active layer of each processing stage of the system perpendicularly to the direction of the signal flow and sandwich a heat sink, made of a beryllia-based ceramic and with high heat conductivity, between the slices.

It will be possible to develop a technique to materialize this structure by modifying the existing multichip carrier mounting technique. An electrooptical converter element may be attached to each end of each slice carrying compare-exchange cells, thereby making up a basic component unit of the processing stage. The processing stage may comprise a desired number of slices.

9. Conclusion

Three-dimensional VLSIs should not merely be substitutes for two-dimensional VLSIs for use when the integration density can no longer be raised for the two-dimensional VLSIs. The introduction of three-dimensional VLSIs can lead the way to a totally new area. Even though the general computation capacity of three-dimensional VLSIs has not been evaluated, it has become increasingly clear that three-dimensional VLSIs may far exceed the two-dimensional ones with respect to some important aspects of their operation. This seems to suggest that the efforts devoted to the development of three-dimensional VLSIs will be adequately rewarded.

References

[BILA84] G. Bilardi and F.P. Preparata, "An Architecture for Bitonic Sorting with Optimal VLSI Performance," IEEE TRANS. COMPT., Vol C-33, No 7, July 1984, pp 646-651.

[CARD86] H.C. Card, W. Pries, and R.D. McLeod, "Contributions to VLSI Computational Complexity Theory From Bounds on Current Density," INTEGRATION, Vol 4, No 2, June 1986, pp 175-183.

[DOHI82] Y. Dohi, A. Suzuki, W. Matsui, "Hardware Sorter and Its Application To Database Machine," Proc. 9th Int'l Symp. Comput. Arch., pp 218-225.

[HASE86b) M. Hasegawa and Y. Shigei, "Sorting on Optically-Connected Three-Dimensional VLSI," the Information Processing Society of Japan, study report 86-CA-63-aa, 1986, pp 99-107.

[HASE87] Ibid., "Tradeoffs Regarding Drive Delay Time Between Optically-Connected Communications and Wire-Connected Communications on VLSIs," JOURNAL OF THE INFORMATION PROCESSING SOCIETY OF JAPAN, Vol J70-D, No 3, 1987, pp 650-651.

[HIBI86] Information provided by Y. Hibino of NTT.

[KNUT73] D.E. Knuth, "The Art of Computer Programming," Vol 3: Sorting and Searching. Reading, MA: Addison-Wesley, 1973.

[KURO86] K. Kurokawa and H. Aiiso, "Three-Dimensional ICs," JOHOSHORI, Vol 27, No 7, July 1986, pp 718-729.

[MEAD82] C. Mead and M. Rem, "Minimum Propagation Delays in VLSI," IEEE J. SOLID-STATE CIRCUITS, Vol SC-17 No 4, August 1982, pp 773-775.

[MURO82] S. Muroga, "VLSI System Design," John Wiley, 1982.

[MORA79] H.P. Moravec, "Fully Interconnecting Multiple Computers With Pipelined Sorting Nets," IEEE TRANS. COMP., Vol C-28, No 10, October 1979, pp 795-798.

[ROSE83] A.L. Rosenberg, "Three-Dimensional VLSI: A Case Study," J. ACM., Vol 30, No 3, July 1983, pp 397-416.

[STON71] H. Stone, "Parallel Processing With the Perfect Shuffle," IEEE TRANS. COMPUT., Vol C-29, February 1971, pp 153-161.

[THOM77] C.D. Thompson and H.T. Kung, "Sorting on a Mesh-Connected Parallel Computer," C. ACM., Vol 20, April 1977, pp 263-271.

[THOM80] C.D. Thompson, "A Complexity Theory for VLSI," PhD dissertation, Carnegie-Mellon University, August 1980.

[THOM83] Ibid., "The VLSI Complexity of Sorting," IEEE TRANS. COMP., Vol C-32 No 12, December 1983, pp 1171-1183.

[VUIL83] J. Vuillemin, "A Combinational Limit to the Computing Power of VLSI Circuits," IEEE TRANS. COMP., Vol C-32, No 3, March 1983, pp 294-300.

20109/9365

Tabular List Representation Oriented Machine

43067589c Tokyo KONPYUTA AKITEKUCHA SHINPOJIUMU RONBUNSHU in Japanese 17-18 May 88 pp 89-97

[Article by Ryoichi Wada, Yutaka Aoki, and Masato Honma, Information System Laboratory, Matsushita Electric Industrial Co., Ltd.: "Architecture of Symbol Processing Machine ATOM Designed for High-Speed List Pattern Matching"]

[Text] In representing a list, the ATOM (a tabular list representation oriented machine) uses a set of leaf data distributed in a tree structure whose end nodes indicate the absolute-location node addresses on the tree. the ATOM processes the leaf data using more than one processor in parallel, making itself capable of handling lists at high speed, in particular, in relational operations. The results of simulations carried out using sample programs have indicated that the ATOM may be able to execute production systems more than 10 times as fast as the conventional machines. This paper deals with the new method of internal list representation, how to process the list representation, the structure of the ATOM, and the results of simulations.

1. Introduction

It is characteristic of the programs used in the field of artificial intelligence to employ a list structure for data, with many of the control structures adopted being of the data drive type.

The use of a data drive-type control structure signifies that a lot of pattern matching is carried out during program execution. In fact, it is well known that pattern matching constitutes a major cause of bottlenecking during program execution in this field.[1]

An explanation of the bottlenecking is as follows: List data is stored in memory in the form of list cells. When lists are compared, all the elements of the corresponding end nodes must be accessed. To perform this process, it is necessary to trace the lists progressively. These procedures cause bottlenecking in memory access.

A processing system designed to make the above-mentioned list tracing easier has been developed. It furthers CDR coding and completely converts the lists to be processed into sequences.[2] One of its drawbacks is that, when processing lists for other purposes than matching, the lists must be converted into an ordinary type.

When the data to be processed consists of more than one element that must be processed in the same manner as list comparison, the processing time can be shortened through parallel processing. However, if, as in the case of list cell representation, the relationships among the elements making up the data are indicated on a relative location basis, it is necessary to access the elements sequentially. In this case, parallel processing is impractical.

The symbol processing machine ATOM is designed to be capable of processing lists at high speed. It uses a tree structure whose end nodes include elements indicating the node addresses, i.e., the absolute addresses in the tree structure. It represents lists using sets of leaf data of the tree structure. It has a structure of the SIMD-type, adopted with attention given to the leaf data independence in a tree structure. The SIMD-type structure enables the leaf data to be processed in parallel using more than one processor. In representing lists, the ATOM retains the semantic attributes of the lists so that they can be processed as they are. This is a characteristic of the list representation made by the ATOM.

According to Pleszkun, et al.,[7] the list representation made by the ATOM is identified with structure-coded list representation. As stated in the reference literature (7), the structure-coded representation permits high-speed access to lists. Therefore, the ATOM also enables high-speed list accessing. In addition, it efficiently supports pattern matching. Up to now, the utilization of the structure-coded representation for LISP machines has not been studied much.[7] The ATOM has not been developed as a LISP machine, but it will become the first real machine to have the capability of list representation introduced above.

In Section 2 of this article, the above-mentioned new type of list representation will be explained. In Section 3, the architecture of the ATOM will be described. Section 4 will deal with the list processing system of the ATOM. In Section 5, the results of simulations will be discussed.

2. List Representation

In the ATOM, a list is represented by a set of leaf data. The leaf data referred to here comprises the element included in an end node to which a node address in the tree structure has been added.

```
<list-data> ::= {<leaf-data>*}
<leaf-data> ::= <node-address>
                <atom>
```

The node address is represented in a vector. When an expression S is represented using a multilevel tree, as shown in Figure 1, the data representing it comprises numbers extracted from different levels of the tree and arranged in ascending level order. Each level of the tree is assigned consecutive numbers arranged in ascending order from left to right.

Take a list comprising

(A (B (C)) D),

for example. It is represented by data,

{ [1]A [2 1]B [2 2 1]C [3]D }

A characteristic of this list representation is that each piece of leaf data is highly independent. This is because, in this list representation, each piece of data has information regarding its own absolute location in the tree structure. Therefore, this list representation is not dependent on the order of the leaf data. For example, the above list can also be written as shown below.

{ [2 1]B [2 2 1]C [3]D [1]A }

It is possible to interpret this list representation as representing a list in the form of a table of node addresses and elements. In this article, the list representation by the ATOM will subsequently be referred to as the tabular list representation.[3]

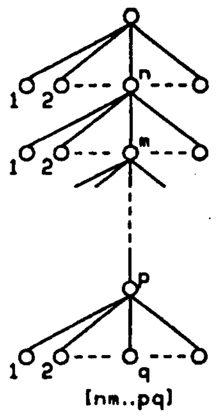


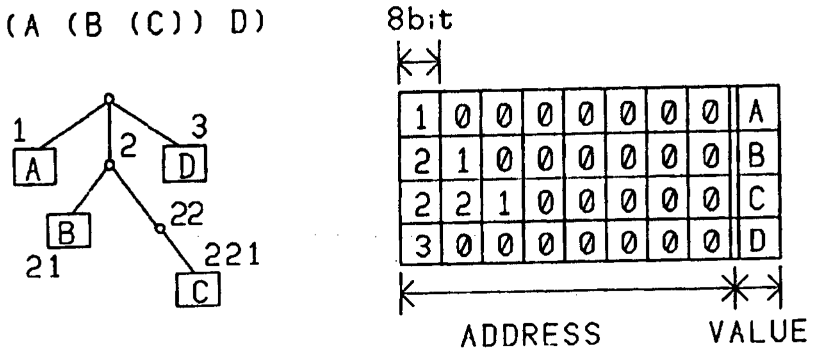
Figure 1. Node Addresses of Leaf Data

The ATOM has a structure incorporating more than one processor to process leaf data (entries in a tabular list representation) in parallel. Since the leaf data have the above-described properties, they can be processed in parallel.

To actually enable the parallel processing of leaf data to occur, it is necessary to set such specifications as the number of leaf data to be processed in parallel and the number of node address bits. According to the analysis of some typical application programs, of the lists that became the objects of processing, those with a length of 255 or more, a depth of 8 or more and 64 or more leaves accounted for 5 percent or less.[3]

Based on the above results, we have set the number of leaf data processes at 64, the node address vector length at 8, and the number size at 8 bits.

A multilevel tree corresponding to the list example above is shown in Figure 2 along with the corresponding tabular list representation. The contents of the address part of the table shown in Figure 2 constitute node address vectors. The size "8 bits x 8" defines the maximum number of subnodes a node can hold and the maximum depth of lists that can be processed. The number of entries in the table, up to 64, corresponds to the maximum number of atoms that can be included in a table. A list that does not comply with these restrictions is treated as an exception to be divided into sublists for exceptional processing.



1	0	0	0	0	0	0	0	A
2	1	0	0	0	0	0	0	B
2	2	1	0	0	0	0	0	C
3	0	0	0	0	0	0	0	D

Figure 2. Tabular List Representation Example

3. Structure of ATOM

As shown in Figure 3, the ATOM consists of one DOU (data operation unit) and more than one SOU (structure operation unit). In the ATOM, it is only the DOU that reads instructions and operates under self-control. The SOUs operate only concomitantly with the DOU. The DOU, like an ordinary computer, consists of a microprogram-type controller, a memory, registers used for data processing, and an ALU (arithmetic and logic unit). In addition, it has an interface for use in exchanging data with the SOUs. The SOUs have identical structures, mainly comprised of a leaf memory for storing leaf data, registers, an ALU, an M flag used in the matching operation, an E flag used to indicate that the SOU has no leaf data, and other flags used to indicate the results of operations performed by the ALU.

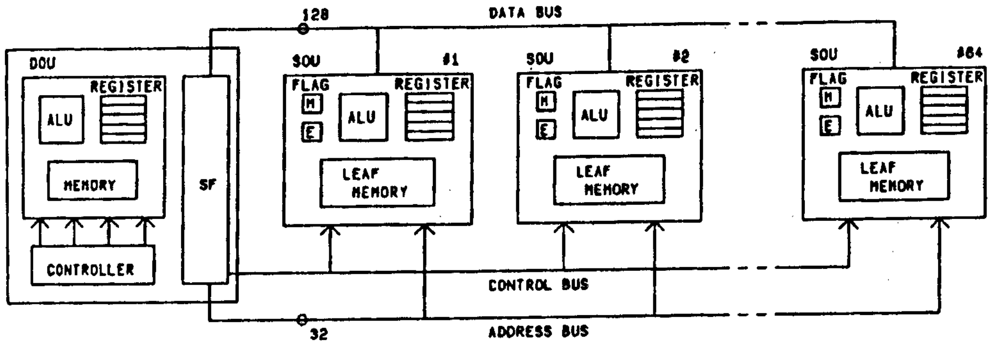


Figure 3. Block Diagram of ATOM

Each word of the data stored in the DOU consists of 40 bits as shown in Figure 4. Of the 40 bits, the high-order 8 bits constitute a data tag and the remaining 32 bits carry actual data. For an atom, the data part contains a pointer or a data value. For a list, it contains an address in the SOU's leaf memory where leaf data included in the list are stored. A set of leaf data making up a list is stored at the same addresses in the leaf memories of different SOUs.

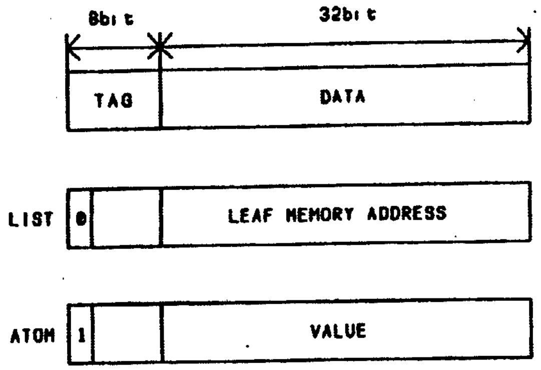


Figure 4. Data Stored in DOU

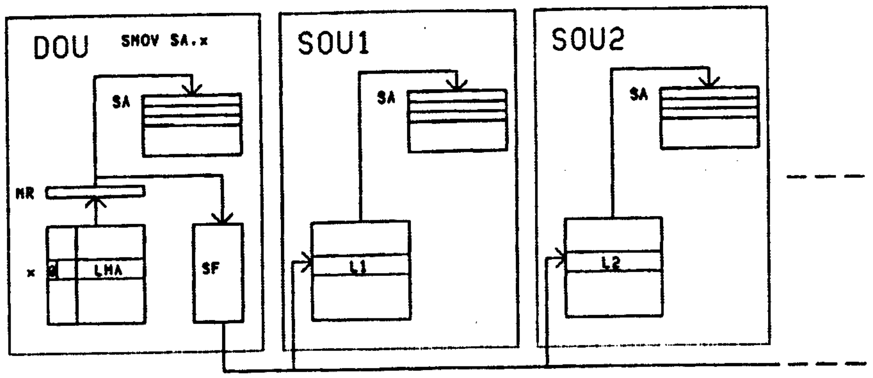


Figure 5. Execution of SMOV Instruction

Figure 5 illustrates the execution of an SMOV instruction issued to transfer data. In the example shown in Figure 5, list data is transferred from the memory to registers. The registers of each SOU are related to specific registers of the DOU. When list data are read from leaf memories, they are stored in SOU registers. The operation performed when an SMOV instruction is executed differs according to whether the target data is an atom or a list. When it is a list, in addition to the DOU operation, the SOUs are also operated as described above. In this case, first the 40 bits of data at the effective address given by the SMOV instruction is read from the DOU memory. Then, in the DOU, the data tag is checked to determine whether the data read out is list data. If it is found to be list data, the DOU sends out the low-order 32 bits of the data to the memory address bus connected to the SOUs. Each SOU then transfers the leaf data stored at the specified address of its leaf memory to its registers. In this way, all the leaf data making up a list are transferred to registers of all the SOUs at the same time. Subsequently, the SOU registers to which the leaf data have been transferred are kept interlocked with the corresponding DOU registers. In other words, normally in the ATOM, once the list data are loaded in the registers, the entire list can be processed. The leaf data composition in each SOU is shown in Figure 6. The value part consisting of 40 bits is identical with that of the data composition in the DOU, but the leaf data does not contain list data.

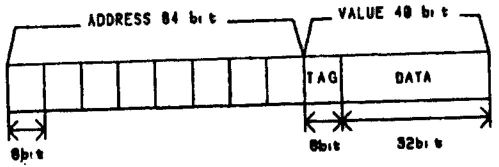


Figure 6. Leaf Data Composition in SOU

The DOU and the SOUs are connected by a 128-bit wide data bus through which leaf data are transferred. This data bus is used, for example, to merge leaf data to link lists or to bring specific leaf data into the DOU.

Table 1. Instruction Set Used by ATOM

Classification	Instruction examples	Selection condition	Trap condition
Leaf processing			
List move	SMOV, SPUSH, SPOP	No	No
Leaf move	GET_LF	Yes	No
Leaf merge	MERGE, APND	No	No
Leaf delete	DEL-LF	Yes	No
Node address operation	ADD_TAD,SUB_TAD,SHD_AD	Yes	Yes
Matching			
Leaf matching	MCH_LF	Yes	Yes
Node address matching	MCH_AD	Yes	Yes
Element matching	MCH_VL	Yes	Yes

Table 1 lists some of the instructions included in the instruction set used by the ATOM in performing various list processing operations. As shown in Table 1, some of the instructions listed permit a selection or trap condition to be specified. A selection condition, if specified, causes only the selected leaf data to be processed. To specify it, the earlier-mentioned M flags of the SOUs or the ALU flags are used. A trap condition can be specified in the same way as a selection condition can. It is used to delete leaf data after the execution of an instruction.

4. List Processing

In this section, the principal list processing operations performed by the ATOM will be explained.

(1) Basic list operation

The basic list operation, such as car, cdr, and cons, can be performed by properly combining leaf-data node address processing, specific leaf-data deletion and leaf-data merging. The cdr operation, for example, for the previously mentioned list (A (B (C)) D) can be performed as shown below.

```
cdr( { [1]A [2 1]B [2 2 1]C [3]D } )
= { [0]A [1 1]B [1 2 1]C [2]D }
= { [1 1]B [1 2 1]C [2]D }
```

In this operation, 1 is subtracted from the heading number of the node address vector for every item of leaf-data, and the leaf-data corresponding to the node address vector that becomes 0 following the subtraction is deleted.

```
1 SMOV SA, X
2 SUB_TAD_TZR SA,1
```

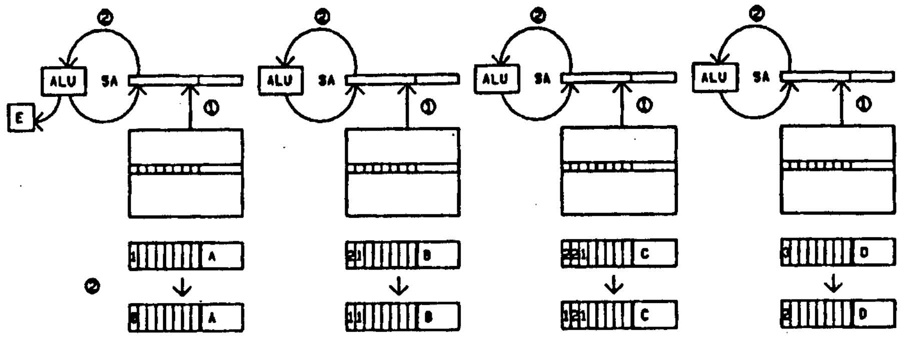


Figure 7. cdr Operation

The processing actually performed in each SOU during the above operation is illustrated in Figure 7. During the process marked 2 in Figure 7, the two operational steps described above are performed simultaneously against the leaf-data stored in the SA register. The leaf-data whose node address vector becomes 0 following subtracted of 1 is trapped and deleted (the trap condition mentioned in Section 3 has been specified). More specifically, the E flag of the specific SOU is set, indicating that the SOU has become empty. As indicated in Table 1, the trap condition can be specified for all the node address operation instructions and also for the match instructions explained later. The SOU having its E flag set does not operate. It is treated as if it stores no data.

As mentioned above, in the DOU, the list and atom data are represented differently. Therefore, when list processing produces atom data, it becomes necessary to change the data format in the DOU. The car operation for the list (A (B (C)) D) is performed as shown below.

```
car ( { [1]A [2 1]B [2 2 1]C [3]D } )
= SHIFT_UP_ADDRESS( { [1]A } ) = { []A }
```

In the above operation, a node address shift-up (the heading data is deleted from the node address vector, reducing the leaf-data depth by one level) occurs and, as a result, only one piece of leaf-data, without any node address, is obtained. In the above case, it becomes necessary to change the data format in the DOU as previously mentioned. The data format is automatically changed via the SOU interface (SF) in the DOU. More specifically, when an element produced as a result of an operation loses its node address, the SF detects it through hardware and the DOU microprogram changes the contents of the corresponding register in the DOU into an atom.

(2) Accessing atoms in specific locations in list

This operation becomes necessary quite often even though, since it is executed using such functions as car and cdr for list processing, its function is not included among the basic ones for list processing. The list processing language pop-11 offers this function.[6] When an ordinary list cell operation method is used, this operation is performed by tracing the target list until the target atom location is reached. The ATOM enables the target atom location in a list to be reached directly. When the ATOM performs this operation, it first outputs the target node address from the DOU to the data bus, providing an instruction for the matching operation executed by the SOUs. As indicated in Table 1 matching can compare a leaf-data node address, leaf-data value or an entire set of list representation in each SOU with the data existing on the data bus. When the data from an SOU is found to match the data present on the data bus, the M flag of the SOU is set. After the M flag of the SOU storing the element to be accessed is set, the DOU can access the SOU directly (by specifying the M flag as the selection condition). Figure 8 illustrates an example of a matching operation.

(3) List Comparison

The equivalence between two lists can be determined based on the following two conditions:

(i) All the leaf-data included in a list are also included in the other list.

(ii) The other list contains no other leaf data.

The second condition has become necessary since the list representation by the ATOM does not include a termination indicator "nil."

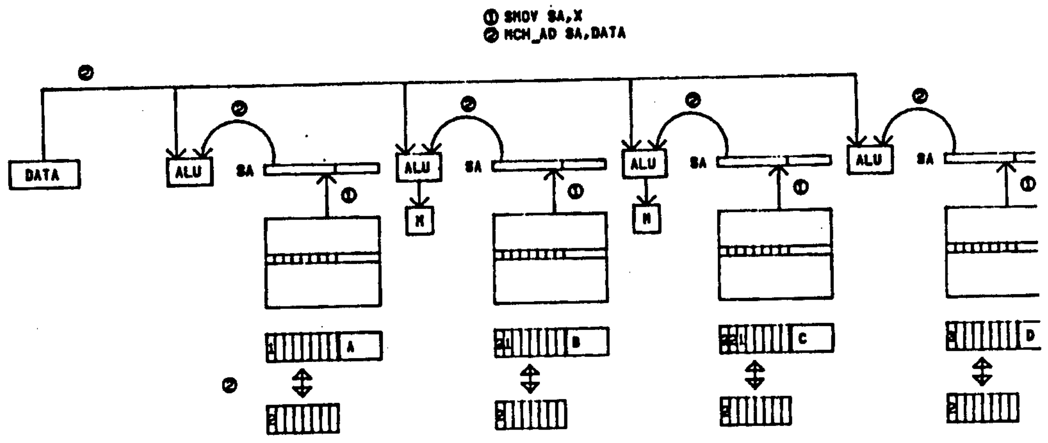


Figure 8. Matching Operation Example

This list comparison can also be performed directly, without involving indirect list processing. When it is performed, first, the two lists to be compared are loaded into registers from memory. Subsequently, the leaf data included in one of the two lists are sequentially read out to the DOU and are matched with the contents of the other list in the above-described manner. The matching leaf data found between the two lists are deleted. The matching operation is performed for all the leaf data included. If the two lists are found to have become empty upon the completion of the matching operation, the two lists are judged to be equivalent.

In the above-described way, the list comparison operation, complex and time-consuming when performed by the conventional method, can be executed through the simple repetition of leaf data comparison by the ATOM.

(4) Pattern matching

Pattern matching becomes necessary when one of the two lists compared for determination of equivalence comprises a pattern. A pattern, as referred to here, is a list containing undefined elements. More specifically, it is a list with some elements represented by variables.

In the ATOM, the variables involved in the matching operation are also assigned a data type. The data type assigned constitutes a piece of atom data identified by a tag. Therefore, a pattern, if containing no segment element, can be compared with a list as described above, except for the binding of open and closed variables. In comparing them, open variables are regarded, at the instructional level, as unconditionally matching whatever value will improve the matching efficiency.

If the pattern contains a segment variable, the element following the segment variable is searched for in the data list, and the elements skipped during this process are bound to the segment variable. When this operation is performed, an environment preservation process required for back-tracking is also executed.

As described above, a segment variable can be processed through the retrieval of the subsequent element. The retrieval is easy. When a segment variable occurs, the ATOM outputs the value part of the next leaf data from the DOU to the data bus and compares the output value with the data store in each SOU.

5. Simulation Results

To verify the effectiveness of the ATOM architecture, we carried out a dynamic characteristic examination and a preliminary performance evaluation using sample programs before analyzing real-use programs on a real machine.

Specifically, we created a simulator, of the machine language level, for collecting such data as the execution frequency of each machine-language instruction and the total number of execution cycles (counted with the addressing mode and data taken into consideration) and executed the following programs using the simulator.

(a) Differential program (Derivative) (4)
(b) Monkey and bananas (5)

The differential program is designed to differentiate the symbols included in simple polynomials. The processing performed by the program mostly comprises list processing, with no pattern matching. Coding is by manual compilation based on the assumption that the LISP program mentioned in the literature[4] is to be executed on the ATOM. For the reader's information, "cons" and function designator are frequently executed in the LISP program.[4]

The Monkey and Bananas program is often used as a production system sample. Most of the processing performed by this program comprises pattern matching. The Rete algorithm[1] is not used in this program. The patterns appearing in this program do not contain segment variables. We carried out coding directly, using the ATOM assembly language, while being somewhat concerned about high-level languages.

(1) Examination of dynamic characteristics

We classified the instructions for use by the ATOM into groups and calculated the ratio, among the instruction groups, of the execution of each group of instructions as well as that of the number of execution cycles counted during their execution. The results are listed in Tables 2 and 3. From the results of the ratio calculations, the following can be said:

1) The instructions for list moving are the highest with regard to the ratios of both execution frequency and the number of execution cycles. It is assumed that they are executed mostly for accessing stacked parameters. Therefore, the ATOM operation will be accelerated further with the adoption of a stack cache.

2) Even in such a program as Monkey and Bananas, that performs mostly pattern matching, the number of execution cycles counted during the execution of matching-related instructions is about 10 percent of the total

Table 2. Instruction Execution Ratios for Differential Program

Classification	Instruction examples	Execution frequency ratio (%)	Execution cycle count ratio (%)
Leaf processing			
List move	SMOV,SPUSH,SPOP	32	37
Leaf move	GET_LF	26	22
Leaf merge	MERGE,APND	26	22
Leaf delete	DEL_LF	26	22
Node address operation	ADD_TAD,SUB_TAD,SHD_AD	26	22
Matching operation			
Leaf matching	MCH_LF	0	0
Node address matching	MCH_AD	0	0
Element matching	MCH_VL	0	0
Tagged-data operation instructions	SADD,SEQ,SETT	3	3
Control instructions	JMP,LOOP,CALL,RET	17	18
Other general instructions	ADD,TEST,DEC	22	20

Table 3. Instruction Execution Ratios for Monkey and Bananas

Classification	instruction examples	Execution frequency ratio (%)	Execution cycle count ratio (%)
Leaf processing			
List move	SMOV,SPUSH,SPOP	25	33
Leaf move	GET_LF	22	19
Leaf merge	MERGE,APND	22	19
Leaf delete	DEL_LF	22	19
Node address operation	ADD_TAD,SUB_TAD,SHD_AD	22	19
Matching operation			
Leaf matching	MCH_LF	10	8
Node address matching	MCH_AD	10	8
Element matching	MCH_VL	10	8
Tagged-data operation Instructions	SADD,SEQ,SETT	1	1
Control instructions	JMP,LOOP,CALL,RET	23	21
Other general instructions	ADD,TEST,DEC	19	18

number of cycles. This low ratio is thought to be attributable to the effect of parallel leaf-data processing, made possible by the ATOM architecture.

3) The execution cycles counted during the execution of the instructions for leaf processing (various instructions are combined to perform basic list processing), excluding those for list moving, account for about 20 percent of the total execution cycles.

4) When the leaf processing, matching operation, and tagged data processing performed by the ATOM are referred to jointly as symbol processing, the ratio of symbol processing to nonsymbol processing, with regard to the number of execution cycles, is 6:4. The non-symbol processing is performed through the execution of control instructions and other general instructions. The number of execution cycles counted during symbol processing, as well as those counted during non-symbol processing, accounts for about 20 percent of the total number of executed cycles.

(2) Comparison with conventional machines

To compare the ATOM with the Symbolics-3600 (Common LISP) and SUN3-260 (pop-11), programming was conducted on the three machines using the respective languages. For the ATOM, the execution time was calculated based on the total number of execution cycles determined by carrying out simulation with the microprogram cycle set to 125 ns (8 MHz). During the sample program simulations, no list requiring the exceptional processing mentioned in Section 2 occurred. The execution times calculated based on the results of sample program simulations are listed in Table 4.

Table 4. Sample-Program Execution Time (in ms)

Sample program	Machine → name	ATOM	SUN (POP-11)	SYMBOLICS (Common LISP)
Differential program		0.43 (1)	2.5 (5.8)	0.76 (1.8)
Monkey and Bananas		11 (1)	250 (23)	1,260 (115)

Note: Figures in ( ) indicate execution times relative to the base figure of 1 for the ATOM.

Even though the execution efficiency achieved by the ATOM in executing the differential program, when compared to the execution time required by the other two machines, was not very great, the improvement exhibited by the ATOM in executing Monkey and Bananas was remarkable, indicating the ATOM's potential for high-speed processing. As mentioned before, the differential program is mostly comprised of list processing, while the contents of processing performed in the Monkey and Bananas mainly include pattern matching. Therefore, the difference in high-speed processing efficiency exhibited can be assumed to have resulted from the fact that the ATOM architecture was more suitable for pattern matching than for list processing.

6. Conclusion

A new tabular list representation that utilizes list cells, differing from the conventional list representation, and the architecture of the symbol processing machine ATOM that has been designed to process the new list representation at high speeds have been described. Simulations were carried out to study the dynamic characteristic of the ATOM and to evaluate its performance preliminarily.

The results of simulations performed using sample programs have indicated that the ATOM, as it has been intended, is noticeably efficient in accelerating the execution of programs, the contents of which mostly comprise pattern matching. Therefore, the ATOM architecture that enables the parallel processing of leaf data has been ascertained to be effective in executing programs at high speed.

At present, ATOM hardware of the bit-slice type for operation under microprogram control is being developed. The microinstructions are of a 128-bit horizontal type. Each consists of fields used to control the DOU and SOUs independently. The leaf memory of each SOU will be able to store 64 K leaf data, and the basic capacity of the DOU memory will be 2 M words. A service processor is linked to the ATOM in order to realize the functions of microprogram loading, debugging, and monitoring.

The future tasks with regard to the ATOM include a performance evaluation through the execution of large-scale programs on the real machine, an adaptability study of algorithms, e.g., Rete, in order to enhance the matching efficiency in production systems, and the materialization of the algorithm judged desirable according to the study results.

References

1. C.L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," ARTIFICIAL INTELLIGENCE, Vol 19, No 1, 1982, pp 17-37.

2. ART Reference Manual, Inference Corp. 1987.

3. R. Wada, et al., "Internal Representation and Processing System in List Structure," the Information Processing Society of Japan, Material Prepared by Symbol Processing Study Group, 39-5, 1986.

4. R.P. Gabriel, "Performance and Evaluation of LISP Systems," the MIT Press, 1985.

5. C.L. Forgy, "The OPS83 User's Manual," Department of Computer Science, Carnegie Mellon University, 1986.

6. R. Barrett, et al., "POP-11: A Practical Language for Artificial Intelligence," Ellis Horwood, 1985.

7. A.R. Pleszkun, et al., "The Architecture of LISP Machines," COMPUTER, Vol 20, No 3, 1987, pp 35-44.

20109/9365

Parallel Processing Architecture, Sensor Information Processing

43067589d Tokyo KONPYUTA AKITEKUCHA SHINPOJIUMU RONBUNSHU in Japanese
17-18 May 88 pp 199-208

[Article by Masatoshi Ishikawa, Product Science Laboratory: "Parallel Processing Architecture for Sensor Information Processing"]

[Text] Parallel processing architecture for realizing intellectual sensors is discussed here. First, the sensor information processing configuration is outlined and classified into two types, i.e., the SIMD-type and MIMD-type. Then, the characteristics of each type and its application field are described. In addition, as concrete prototype systems, 1) the prototype parallel processing chip (SPE-8) and its application to a tactile sensor (SIMD), 2) the designing of large-scale parallel processing, and 3) a passive-type sensor system which uses a transputer, are described. Problems associated with processing architecture in realizing sensor fusion and neuro computing are investigated from the aspect of sensor information processing.

1. Introduction

Among the various automated devices which comprise an intelligent robot, sensors hold the key to their high performance functions and mechanisms. Recently, along with the progress in the higher integration of electronic circuits, the realization of the concept of sensor intellectualization is in great demand. With this concept, sensors are not recognized as simple signal conversion devices as they have been in the past. Instead, they are thought of as information processing modules by conducting the sensor's inherent processing using a subsistent operation processing mechanism in combination with detection functions. When this operation processing mechanism is viewed from the processing system, i.e., the entire computer system, it seems to be part of a parallel distributed structure. Unfortunately, however, parallel architecture is not currently well recognized.

Accordingly, when taking sensors applied to an intelligent robot as an example, the sensor's processing configuration is divided into two categories. One includes the processing of information from high homogeneous sensors centered around pattern processing, and the other is the processing of information from heterogeneous sensors. Suitable architecture for each processing configuration will be shown, and their comparisons made.

First, regarding parallel processing architecture of the SIMD structure that is suitable for the former's configuration, the LSI architecture for parallel processing whose prototype was constructed to realize local pattern processing is shown. The example of a tactile sensor combined with the detection unit is described. In addition, a plan to realize large-scale parallel processing by the use of multiple LSIs is outlined.

Then, an experimental language to describe the processing structure of a sensor is discussed. From the viewpoint of sensor fusion which integrates and unites various kinds of sensor information, the architecture mentioned above is arranged, and a technique for its realization is proposed.

In addition, the positioning of sensor information as part of the realization of a computation structure based on the neuron network model and method to realize the processing structure, which have become the subjects of study in recent years, are also discussed.

The architecture discussed in this paper is able to connect sensors, i.e., I/O directly with each CPU. This is due to the attempt to suppress the I/O bottleneck under parallel processing. In this respect, it is the architecture that is believed to regard the I/O as important. Particularly, since the I/O formation plays an important role, this architecture is indispensable in realizing the neuron network model.

2. Parallel Processing in Sensor Information Processing

2.1. Intellectualization of sensors

Prior to discussing the definite processing structure, a concept[1] regarding the intellectualization of sensors is outlined as it relates to the entire system, and is then organized.

The recent advances in the integration technology focusing on semiconductors has brought small and high performance devices of various kinds based on the LSI technology. Due to improvements in the degree of integration, the basic design concept of a circuit and a computer system has been drastically changed. Namely, such design standards as optimization, high speed and minimization, which were taken into consideration in the past, have disappeared. Instead of these, the possibility, simplicity and flexibility of integration and also the existence of CAD are currently accepted as the design standards. In addition, as the degree of integration increases, the fact that communications costs are becoming higher than the computation costs has been suggested a problem mainly associated with the complicated wiring inside the LSI.[2] However, this involved not only the problems related to the LSI itself, but also those related to the entire computer system. In this respect, distributed processing appears as one means of solving the problems, i.e., under the concept in which local processing is carried out by a local processing mechanism, the idea of "transmittance after processing" rather than "transmittance before processing" is attempted.

This idea can be applied to the processing of sensor information without making any changes. In this case, the existence of essential communications limits the sensor's output. This differs from the computer system, as does the problem involving cost. A communications path between a sensor and the central processing mechanism has a limited number of wires and communications capacity. This could limit sensor applications. In order to solve this problem, it is necessary to improve the quality of the data that flows in the communications path in order to utilize the limited communications capacity effectively. To do so, it is necessary to join the processing mechanism with a detector. This means that a sensor should not be recognized as a simple detection converter, but instead, should be thought of as an independent signal or information processing module. This is called a sensor's intellectualization.

2.2 Concept of Sensor Processing

Following is a summary of the sensor processing concept. Processing on a sensor must be captured through a broader concept than that of ordinary processing. Figure 1 explains this idea. As shown in this figure, when x is thought of as a phenomenon involving measurements or the state of the system Ω, a measured quantity $\bar{x}$ forms a part of the state x, and is converted to that sensor's output s through the sensor's detection conversion mechanism. This process can be expressed by equation (1), where ω is a physical phenomenon that rules Ω, and is used as an operator.

$$S = \omega(x) \tag{1}$$

The ideal mechanism of the sensor is obtained when the operator $\bar{\omega}$, with respect to the measured quantity $\bar{x}$, becomes equation (2) as part of ω,

$$s = \bar{\omega}(\bar{x}) \tag{2}$$

provided that $\bar{\omega}$ is approximately analytical, i.e., if the inverse operation $\bar{\omega}^{-1}$ can be realized (linear is preferable), it is acceptable. In other words, it is acceptable if x is selected from x. In this regard, it is said that a sensor is the device used to extract and select information. However, if the sensor's output s itself cannot realize an ideal sensor, an additional operator σ is applied to s.

$$s' = \sigma(s) = \sigma\bar{\omega}(\bar{x}) \tag{3}$$

σ is designed so that the quantity $\sigma\bar{\omega}$ becomes approximately analytical. This means that the processing of σ by a sensor involves only the expansion of the inherent sensor's mechanism $\bar{\omega}$. Instead of simple processing of σ, the more essential phenomenon ω and/or processing of the characteristics ($\bar{\omega}$) of the sensor itself must be taken into consideration along with their combinations.

Therefore, since physical and chemical quantities, including electric quantities through the sensor characteristics, are dealt with, the realization of a processing structure which utilizes these characteristics is thought to be possible. Based upon this way of thinking, the author

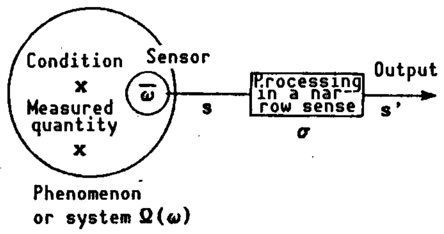

Condition
x
Measured quantity
x
Sensor
$\bar{\omega}$
s
Processing in a narrow sense
σ
Output
s'
Phenomenon or system $\Omega(\omega)$

Figure 1. Concept of Processing on a Sensor

developed a sensor[3] to extract the center of the pressure distribution by using a pressure sensitive conductive rubber, a sensor [4] to display pressure distribution using a pressure sensitive conductive rubber and liquid crystal, and an association memory system[5] for use with optical operation. These devices are designed to realize parallel computation as a physical phenomenon, instead of as an electronic circuit. In this respect, they comprise a kind of parallel processing.

However, the realization of this processing depends on the characteristics of an object. Since the sensors are usually for exclusive use, a limit is placed on their application in most cases. The realization of general-purpose processing is not easy. In this situation, the processing used for specific purposes is not going to be discussed. Instead, the processing architecture for general purposes is proposed through the introduction of processors with different configurations.

2.3 Mechanism and Configuration of Sensor Information Processing

First, the processing mechanism of a sensor, i.e., the mechanism and functions of σ mentioned above, is classified, and examples of concrete operations are presented. Table 1 summarizes these sensor classifications.

Table 1. Classification of Sensor Processing Mechanisms

Classification	Mechanism	Operation example
Correction	Compensation, correction, noise elimination linearization	Average, comparison, filtering
Operation	Abstraction of features, conversion, learning, transformation, integration, recognition	Correlation, average, centroid, sum, moment, comparison
Control	Execution of measurement algorithm, passive sensing	Servo, sequence control, scanning
Transmittance	Output conversion, standardization, compression, modulation	Coding, encoding, error detection/modification
Display	Distribution display, density conversion, visualization	Correlation, scanning, image abstraction
Handling	Instruction execution	Parameter adjustment, sampling order

Of these mechanisms, when some of them are strongly and/or directly related to processing peculiar to a sensor, the effectiveness of the sensor's intellectualization can be expected to a great extent.

Toward such intellectualization, many attempts have been made to unify the peripheral circuit with a detection unit on the semiconductor pressure sensor. One having a built-in scanning circuit has already been made.[6] In addition to these, a sensor[7] designed to unify a scanning circuit which converts the pressure distribution (64 x 64) into video signals utilizing pressure sensitive conductive rubber, a technique to obtain the center of the pressure distribution under a matrix configuration utilizing a parallel processing circuit, and the passive sensing method which would recognize the outline of an object by conducting positioning control of patterns with the above-mentioned circuit are also proposed.[9] The integration of peripheral circuits becomes a very important subject for these sensors when realizing the detection structure corresponding to each sensor.

However, the method utilizing the physical structure, as described above, and/or intellectualization by electronic circuits are for specific usage, not for general purpose use. Therefore, in order to realize processing for more general purpose use, the introduction of processors under some configuration is inevitable.

In the face of the development of intellectual sensors to be used with processors, advances in distributed processing for the entire system using sensors is highly anticipated. Namely, it is necessary to make the positioning of the entire system a staged multiprocessing system. From the top-ranked processor to the bottom-ranked sensor or actuator, through each rank of level, a processor responsible for inherent processing at each level should be realized. This is the idea suggested.

In this paper, the processing of a sensor located in the bottom rank is discussed. In other words, an attempt is made to get the processor to conduct inherent processing for a sensor and to place the processor executing parallel processing inside a sensor.

2.4 Parallel Processing of Sensor Information

The structure of parallel processing involving sensor information is generally divided into two classes, and their characteristics are discussed.

The first configuration corresponds to processing information from visual or tactile sensors which are placed on a matrix or array. The main purpose of this kind of processing involves pattern processing focused on correlative computation. It is suitable for SIMD-type processing.

Regarding the sensor that corresponds to the above configuration, Raibert, et al., are studying a prototype tactile sensor which involves LSI application, called the VLSI tactile sensor.[10] In this sensor, a metal electrode is placed directly on 6 x 3 parallel processors arranged on a silicon wafer, and the pressure sensitive conductive rubber placed on it is used as the pressure sensitive material. Then, a correlative computation of the pressure distribution information is executed. This is the purpose of the sensor. In this paper, the processing structure of this sensor is discussed in Chapters 3 and 4.

The second configuration involves the processing of information from heterogeneous sensors. Since a lesser degree of homogeneousness exists in this processing configuration, this sensor is suitable for MIMD-type processing. With regard to this kind of sensor, Henderson, et al., proposed the concept of a logical sensor.[11] The function of this device is to realize a flexible processing network by defining the logical sensor of the abstract data-type as a processing module, in order to avoid the confusion related to the software accompanying the sensor's intellectualization. That is, the processing of heterogeneous sensors can be broken down into a processing module peculiar to each sensor. An intellectual sensor results from making hardware using the processing sensor. Its processing structure is described in Chapters 5 and 6 in this paper.

3. Local Pattern Processing LSI "SPE-8" and Intellectual Tactile Sensor

First, an outline of the LSI pattern processing used to execute local parallel operations and its actual application to a tactile sensor will be discussed as an example of an intellectual sensor conducting SIMD-type processing.

3.1 Design Concept of Local Pattern Processing LSI "SPE-8"

Using parallel processing LSI to execute local pattern processing was developed first. This LSI is called "SPE-8 (sensory processing element-8)" since eight processing elements corresponding to a sensor are placed on one chip.

In designing the SPE-8, the realization of scanning parallel processing and perfect parallel processing was assumed to form the structure of the sensor using the SPE-8. The realization of scanning parallel processing is discussed in Section 3.4, and that of perfect parallel processing is discussed in Chapter 4.

The objective of designing the LSI and sensor is to realize high integration and high speed while maintaining the general-purpose characteristics. In order to maintain these characteristics to some extent, the sensor was separated from the processing portion, and a structure making the cascade connection possible was adopted in order to correspond to differences in kinds of sensors and the number of detection points. In addition, in order to promote high integration and high speed, a method to input information directly from the sensor into the processing elements was used. Furthermore, since bit serial operation was adopted as the inner operation method, a circuit was constructed according to the limited number of gates. Functional degradation was incorporated through the increased parallelism, and the composition of a transmitting circuit and scanning circuit was also taken into consideration. In addition, the application of a compact circuit to the sensor was attempted.

3.2 Structure of SPE-8

As described above, the SPE-8 is a pattern processing LSI with eight built-in processing elements. Its structure is shown in Figure 2. The eight processing elements employ the 1 x 8 composition. Each processing element has one channel for input and four pairs of connecting wires in four different directions. There is a common instruction among the processing elements, of a 10-bit composition (op code 2 bit + substantial field 8 bit).

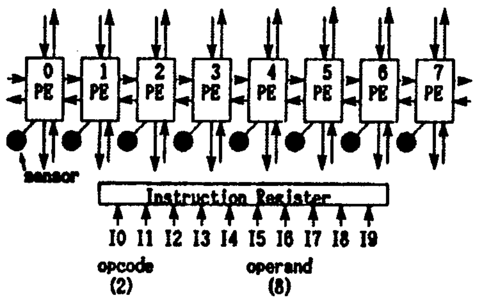


Figure 2. Structure of SPE-8

3.3 Structure of Processing Element

The structure of the processing elements in SPE-8 is shown in Figure 3. Since SPE-8 is intended to be a general-purpose LSI for pattern processing, a multiplication/addition operation function enabling correlative operation is provided in addition to general logical operations and addition/subtraction operations. Moreover, because bit serial operation is adopted, 1-bit, 4-bit, and 8-bit operation is possible.

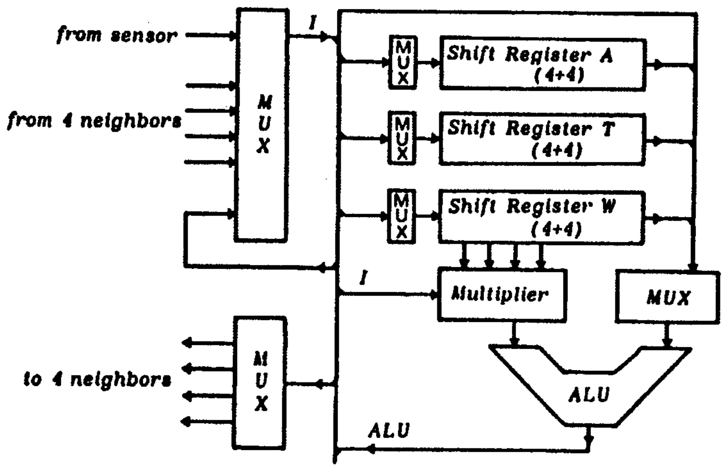


Figure 3. Structure of Processing Element

There are three shift registers inside the processing element, respectively called A (accumulator), T (template), and W (weight). Each register is divided into the upper ranked 4 bits and lower ranked 4 bits. Swapping of each 4 bit [segment] is possible within a register. Only the upper ranked

4 bits on the W-register can be utilized as the multiplication substitute. Data with a 1-bit length corresponds to on-off patterns. Since a register has 8 bits, the processing of 8 data items per register is possible.

Input from the sensor is diversified (becomes multiplex) along with information from neighbors and registers through a Schmidt trigger, becoming I-information as shown in Figure 3.

As a multiplication device to realize multiplication/addition operations, an improvement by reducing the number of gates takes place for Booth's algorithm, currently in general use. Due to this, 4 bit (W-register's upper ranked 4 bit/parallel) x 4 bit (I-data 4 bit/serial) $\rightarrow$ 8 bit (8 bit/serial: input of ALU) multiplication can be realized by a compact circuit.

At ALU, bit serial operation, i.e., serial operation beginning with the lower ranked bit, in order, is carried out. The precise control of the carry bit makes variable data operation possible. There are four different operations--AND, OR, EXOR (exclusive OR) and addition. The handling of negative numbers is possible at an input of the B-side due to the treatment of a sign bit. Regarding the handling of the sign and carry bit, the upper ranked 4 bits (same as 8 bit handling) and lower ranked 4 bits can be used.

An output is chosen from an ALU output, while A and T register outputs and I data. The reason for using I is to permit input data to pass through. It compensates for a 4-neighbor connection, i.e., neighboring data in the oblique direction of 8 neighbors is connected by two sets of 4-neighbor connections.

There are actually 32 control bits inside the SPE-8. However, its composition is of the intensive form of 4 kinds of 10-bit sections, classified according to the execution formation. Since this instructional data is mostly stored by instruction registers, the conditions are maintained without instructional change.

Regarding the control of the SPE-8 interior, synchronizing control using a clock pulse (CP) is applied. A maximum four-stage pipeline structure is adopted inside each cell.

Two different kinds of instruction control methods are assumed. One is the method employing a microprogram involving inherent memory of firmware. The other is the method utilizing an I/O processor by connecting the I/O with a computer. Since subroutine calls and various conditional judgments can be accomplished with the microprogram control method, high level intellectualization is achievable. In this case, however, the scale of the circuit becomes larger. This means that it is not advantageous to reduce the size. In the case of the I/O processor method, reducing the size is easy, but achieving high speed is difficult.

The general specifications for the SPE-8 are shown in Table 2. As seen in this table, the SPE-8 adopts the gate array of 3312 gate integration at its maximum. Its actual integration is 2970 gates. This function can be

Table 2. Specifications of SPE-8

Items	Specifications
Degree of integration	MAX: 3312 gates/chip (2 input NAND conversion) Real: 2970 gates/chip (2 input NAND conversion) = 337 gates/cell x 8 + 274 gates/chip
Delay time	2 ns/gate (inner gate F/O=3, wiring length 3 mm) 3 ns (input buffer F/O=3, wiring length 3 mm) 10 ns (output buffer C_L=15 pF)
Power supply	5 V + 5 percent single power supply
Input/output	TTL compatible
External shape	80-pin, flat package

realized by 337 gates per processing element, while 274 gates are necessary for the normal part.

A maximum execution speed of 87 ns (loading capacity 30 pF) was attained as the result of the delay simulation of a critical path. The actual measured result on the test circuit (memory cycle time: 55 ns) was a 90 ns cycle time, including a delay on a peripheral circuit.

All functions of the SPE-8 were confirmed through software simulation with CAD, tests employing a LSI tester and through the use of an actual circuit.

3.4 Intellectual Tactile Sensor

An example of the intellectualization of a tactile sensor employing LSI will be discussed. This intellectual tactile sensor is able to realize local pattern processing with a high speed when combined with a detection unit (8 x 8) that uses pressure sensitive conductive rubber.

The characteristics of the processing when a tactile sensor is used are: 1) the amount of detectable information is not as great as that of vision, and the localization of the processing is strong; 2) it is necessary to deal with local pattern information; 3) high-speed processing is demanded since results are sent directly to an actuator system as feedback; 4) the sensor is usually installed on the arm and hand of a robot. Therefore, small size and light weight, including compact wiring, are necessary. Due to these features, the effectiveness of the unification of operation' circuits and signal transmittance circuits with a detection unit is believed to be considerable,[14,15] and its development is eagerly awaited.

An example of the composition of the tactile sensor with the SPE-8 is as follows. Figure 1 shows its circuit structure, and Photograph 1 [not reproduced] shows the prototype sensor. As shown in Figure 1, the sensors placed on a matrix are scanned in one direction, and one processing element is allocated to one row. Due to scanning, its processing time becomes longer than that of the perfectly parallel processing method. Simultaneously measurement is not possible with this method. However, its practical processing time is sufficient, as will be mentioned later. Regarding integration, this method has an advantage in that one SPE-8 can handle 64 (8 x 8) sensors. With regard to pressure information, a change in the resistance of pressure sensitive conductive rubber is detected by an electrode. However, since 1-bit operation is used in this case, only on-off information is processed. Because the register of the SPE-8 is 8 bits, each cell can handle only 8 points. When more than 8 points are needed, cascade connections among SPE-8s are possible.

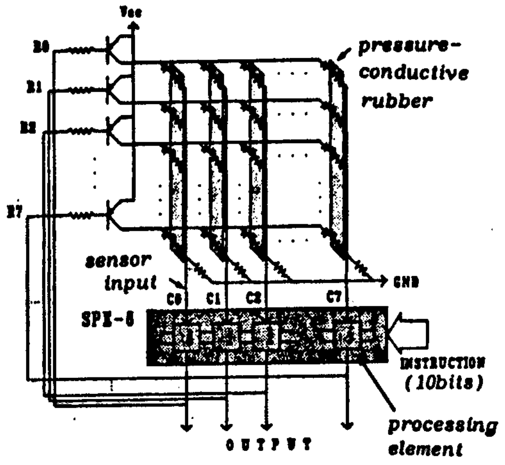


Figure 4. Circuit Composition of Tactile Sensor With SPE-8

As seen in Figure 4, a diode is placed at each detection point. Pressure sensitive conductive rubber is cut into strips, and nonconductive rubber of the same thickness is inserted into that rubber. This prevents the flow of nondirect current and leakage among electrodes, which is peculiar to network resistance.

Instructional control is achieved by sending a program in memory to the SPE-8 in order. With this method, conditional judgment is not possible. However, the confirmation of basic operations can be carried out sufficiently since its requirement for simple pattern processing operations is low.

An actual processing execution example using this sensor is shown. In this example, the edge pattern of an object becomes an output. A concrete operation execution is:

$$\text{output} = (-(q_{i-1,j} \times q_{i+1,j} \times q_{i,j-1} \times q_{i,j+1})) \times q_{ij}$$

This operation only involves 4 neighbors, and is relatively simple. When executing this operation, 12 steps (the establishment of a mode and the set of scanning data) are initially required. Once the initial setup is completed, actual operation is executed. This portion requires approximately 83 steps due to the sensor's response delay. When the cycle time is 100 ns, the operation processing results are obtained as the sensor output at intervals of 8.3 μs.

4. Large-Scale Parallel Processing Using SPE-8

The SPE-8 can realize large-scale processing since mutual connection is possible. Therefore, the design concept and fundamental structure of the architecture (called "SPE-4k") for realizing the perfectly parallel-type sensor information processing will be described. With this system, the processing of information sent from 4096 sensors is directly connected to 4096 processing elements, utilizing 512 SPE-8 units.

4.1 Design Concept

Regarding perfectly parallel-type processing, many systems focusing on graphics processing have been proposed.[16] The connection machine is of this type. The advantage of this type of processing involves its high speed. On the other hand, however, its disadvantages include less flexibility in SIMD-type control, the weakness of neighboring connections, large-scale circuits and bottlenecks in the I/O.

Fortunately, a reexamination of this architecture from the viewpoint of the sensor's intellectualization revealed that these defects can be avoided by allocating a small-scale processing element to one sensor (I/O oriented). This means that, of the defects mentioned above,less flexibility in the SIMD-type control and the weakness of neighboring connections are not significant problems when the degree of exclusiveness of the sensor's processing is considered. The fear of getting large-scale circuits can be diminished through the utilization of small-scale processing elements, such as the SPE-8. Its size can be reduced to equivalent to or smaller than the size of a sensor. Bottlenecks in the I/O will not become an essential problem since the sensors are connected directly to the processing elements.

The conventional system regards the operational mechanism as an important design consideration. Namely, it is designed under the concept whereby the operational mechanism architecture comes first, and then the I/O is connected to it. However, in this system the I/O, i.e., a sensor, is structured first, and then the operational mechanism is added.

4.2 Basic Architecture

The perfectly parallel processing intellectual sensor SPE-4k essentially conducts the processing of homogeneous sensor information from 64 x 64 = 4096 points. The entire architecture is shown in Figure 5. As seen in this figure, SPE-4k is composed of 512 SPE-8 units. In this kind of perfectly parallel method, high-speed processing is possible, and no problems associated with the time lag accompanying scanning exist. The control method and connection with a sensor (optical sensor is assumed) are the same as those of the intellectual tactile sensor mentioned above. However, the integration of the whole (corresponding to 8 sensors by an 80-pin flat package) is not very good due to the SPE-8 package. In addition, SPE-4k is designed to be applicable not only to single-layer connections, but also to

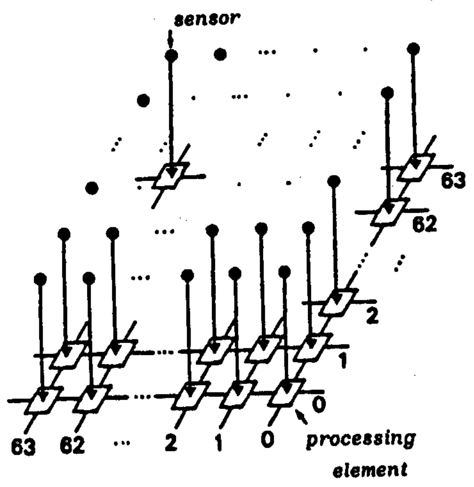


Figure 5. Composition of SPE-4k

multiple-layer composition (4 layers in this case) with the decomposition of 32 x 32 modules.

Detailed designing of the SPE-4k is currently in progress, with its prototype scheduled to be constructed in the very near future.

5. Passive Sensor System by Parallel Processing Using Transputer

The architecture described above is the processing configuration for similar kinds of sensor groups. However, many kinds of sensors are used in the fields of FA and robots. Accordingly, as for the processing architecture of the MIMD-type used to process information from various kinds of sensors, its design concept and structure will be discussed.

5.1 Design Concept

The information obtained from different kinds of sensors has different properties. Moreover, it has different characteristics and configurations. Since it is not effective for a single processing mechanism to process different kinds of information in an intensive manner and, also, with the processing uniformity of this method being bad, the MIMD-type processing configuration becomes effective. However, even in this case, it is necessary to realize the compact configuration of operation mechanisms, such as the CPU and memory, as much as possible since it ultimately aims at the sensor's intellectualization.

On the other hand, not only intrinsic processing functions, but other functions as well can be realized through the local application of general-purpose processors. Of them, the realization of control functions presents the idea of passive sensing as its fundamental concept, and is helpful in improving the quality and quantity of the information obtained from sensors. The idea of passive sensing is based on feedback control between sensors and actuators, and tries to construct a recognition mechanism as its upper-ranked concept. It also attempts to realize a real-time measurement control system with an advanced recognition mechanism.

5.2 Basic Architecture

In order to realize this MIMD-type processing, the prototype information processing system composed of various kinds of sensors, termed the passive sensor system, has been constructed using a transputer that is a CPU suitable for parallel processing. Figure 6 presents a block diagram of its composition.

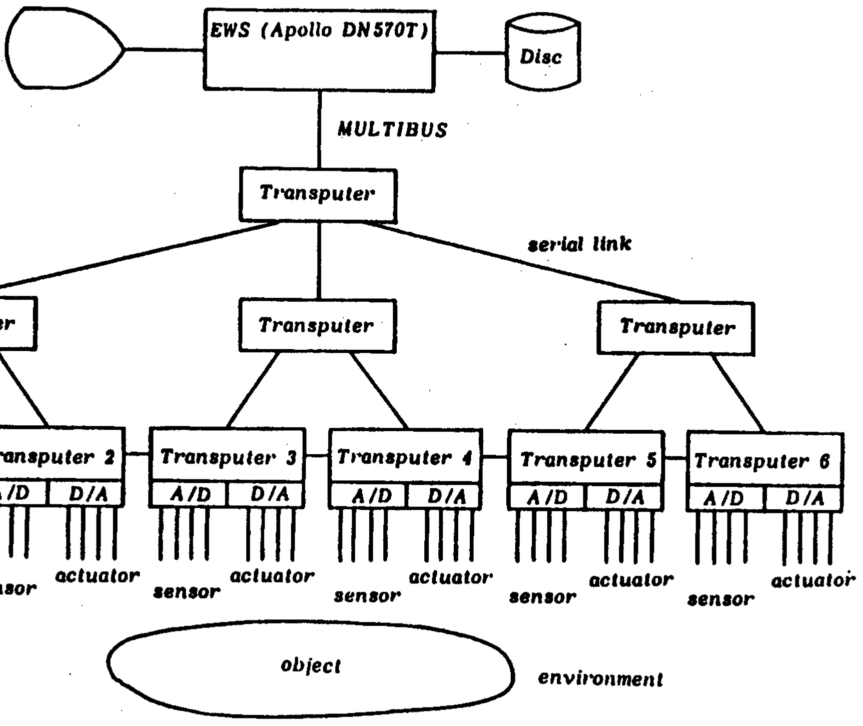


Figure 6. Passive Sensor System

The transputer is a 32-bit processor based upon the RISC concept. Its structure is appropriate for high speed parallel processing based on real-time processing. In developing a parallel processing program, the exclusive parallel processing descriptive language "Occam" is provided. Therefore, advanced parallel processing descriptive ability can be obtained. In addition, since communications and synchronization mechanisms for parallel processes are provided as hardware, problems involving the synchronization of multiprocessor time do not readily occur.

With this passive sensor system, two kinds of boards and 10 transputer units are used in all. Among them, six units are directly connected to sensors and actuators through a 4-channel D/A converter (for input of information from the sensors) and a 4-channel A/D converter (for output to the actuators). One transputer, one A/D converter and one D/A converter are placed in one board. The remaining four units, which communicate with an EWS in a higher rank and conduct processing among sensors and general processing, are located on one board. Communication between CPUs occurs through the inherent serial link of the transputer, and communication with an EWS is carried out with a multibus. Therefore, due to the signal processing on the DWS side, a mechanism similar to parallel processing is realized. Nevertheless, one objective is to realize this mechanism by real-time processing on the transputer side as much as possible.

The manufacturing of this system has been completed, and its performance evaluation and the development of application programs are currently in progress.

6. Sensor Fusion

The object of the passive sensor system mentioned above is the realization of sensor fusion. Its concept is as follows:

6.1 Basic Concept

As the number and kinds of sensors used in a system increase, the necessity for the integration and fusion of information from these sensors increases. The realization of a polymodal processing system in a heterogeneous manner is sensor fusion.[18] Examples include binocular fusion to extract three-dimensional information from visual information through both eyes, and fusion discrepancies between visual and tactile information can be pointed out.

6.2 Processing Software

In order for such a processing system to be realized, the processing structure must be arranged not only by hardware, but also software.

In this regard, the author and others developed experimental language to describe sensor-related processing structures, in a unified manner, from the aspect of software.[19] This language adopted Henderson's concept of the logical sensor.[11] As shown in Figure 7, the logical sensor denotes that operation processing by software or hardware is added to the output (or outputs) of a physical or logical sensor. In addition, the concept of the logical sensor is further advanced and is recognized as the language that includes the memory and learning functions related to sensor information.

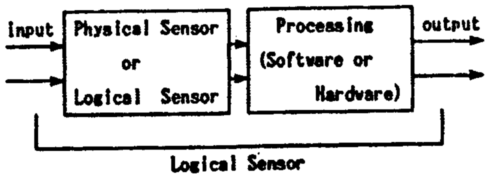

input
Physical Sensor or Logical Sensor
Processing (Software or Hardware)
output
Logical Sensor

Figure 7. Definition of Logical Sensor

Some attempts to simplify the description of sensor information processing by combining the processing on the intellectual sensor and the software, and to realize a smooth conversion toward the intellectual sensor are included in this language. However, since it is experimental language on a simple processor, the following problems exist: 1) When shifted to parallel processing, the load distribution cannot be determined at down-loading; 2) the synchronization time is uncertain. In the case of the passive sensor system mentioned above, the parallel processing descriptive language Occam attached to the transputer and the development system are provided. Regarding the first problem, it could be partially resolved through the use of Occam and the development system functions. With regard to the second, it can be resolved sufficiently with the high-speed task switch function of the transputer.

7. Neuro Computing and Sensor Information Processing

Very recently, parallel processing architecture based on a neuro circuit net model[20] has been attracting attention. From the viewpoint of sensor information processing, the architecture which realizes neuro computing focuses on the I/O structure. The positioning of sensor information and its processing method under neuro computing will be investigated as follows:

7.1 Existence of I/O Bottleneck

When the computation structure of neuro computing is explored by conventional computers, emphasis is placed on realizing its parallel computation structure. This involves the resolution of the conventional von Neumann bottleneck. However, neuro computing is basically an "input → operations → output" device with an advanced parallel structure. In addition, because its ability for operations and memories is low, a large number of processing elements lacking flexibility are placed in a parallel manner, and parallel operations are carried out through a high density network, a network bottleneck must not exist. For this reason, realization through optical operations is also being attempted.

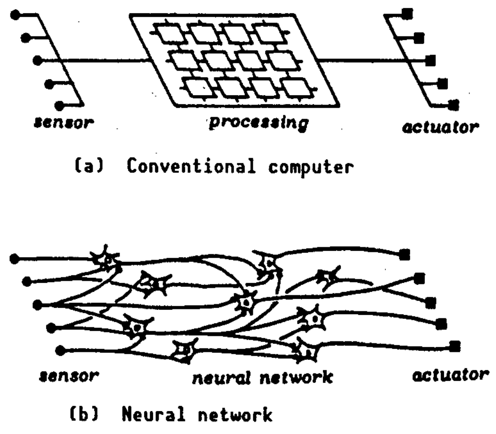


Figure 8. Neuro-Computing and Sensor Information Processing

In order to realize this operational structure without the bottleneck, each element must independently have the "input → operation (specific purpose and less performance acceptable) → output" function. This aspect is compared with that of a conventional computer in Figure 8. The achievement of a high performance operational unit has been targeted in a conventional computer (parallel computer included). A demand for parallelism in an I/O unit has not appeared. Therefore, when attempting to design neuro computing without changing the conventional configuration, a bottleneck occurs in the I/O unit, even though the bottleneck does not appear in operation processing.

7.2 Necessity for I/O Oriented Processing Mechanism

Due to the above reasons, the structure of the processing mechanism realizing neuro computing is very similar to that of a parallel processing-type sensor's intellectualization. This means that even in the case of neuro computing, attention is paid to a network. In particular, the structure of the processing mechanism with the sensor and actuator, as shown in this paper, becomes necessary for its realization.

8. Conclusion

From the viewpoint of sensor intellectualization, operational processing architecture related to sensor information has been discussed.

In this paper, aiming at incorporating a highly general-purpose processing mechanism with a sensor device for exclusive usage, several architectures, including their characteristics, have been described. The actual prototype models and processing results have been shown for some of them.

The operation processing architecture related to sensor information processing has unique design conditions involving integration and the degree of exclusive use when compared to the conventional computer architecture design concept. In this paper, the composition of sensor-oriented operation architecture is basically proposed as the design concept for these conditions. This concept involves paying attention to the I/O portion or unit and then assigning priority to its functions, although the design concept and evaluation of the conventional computer architecture have concentrated on the operation portion or unit. On the other hand, it is pointed out that the architecture realizing neuro computing should have a structure which regards parallelism of the I/O as being important. Therefore, in designing operation architecture to realize neuro computing, the sensor's intellectualization and its design concept have been shown to be necessary.

Along with the higher integration of a computer and/or a processor and the expansion of its field of application, not only the architecture of the computer itself, but also discussions of the architecture, including its applications, should be promoted in the future. The author feels strongly that discussions involving sensor architecture based on the computer architecture described in this paper must become more active.

References

1. Yamazaki, "Sensing Technology II-Sensor's Intellectualization," MEASUREMENTS & CONTROL, Vol 24, No 12, 1985, pp 1135-1142.

2. H.J. Siegel, "Interconnection Networks for Parallel and Distributed Processing," An Overview, IEEE TRANS. COMPUTERS, Vol C-30, No 4, 1981, pp 245-264.

3. Ishikawa and Shimojo, "A Method To Detect the Center of Two-Dimensional Load With Use of Pressure Sensitive Conductive Rubber," J. OF MEASUREMENT & AUTOMATIC CONTROL SOCIETY, Vol 18, No 7, 1982, pp 730-735.

4. Shimojo and Ishikawa, "Pressure Distribution Display Method by Pressure Sensitive Conductive Rubber and Liquid Crystal," J. OF MEASUREMENT & AUTOMATIC CONTROL SOCIETY, Vol 21, No 2, 1982, pp 177-182.

5. M. Ishikawa, et al., "Optical Association--A Simple Model for Optical Associative Memory," APPLIED OPTICS, submitted.

6. C. Kowalski, "Silicon Sensors for Tactile Arrays and Distributed Touch Sensing," SME TECHNICAL PAPERS, MS84-1040, 1984.

7. Ishikawa and Shimojo, "Pressure Distribution Sensor With Video Signal Output and Tactile Pattern Processing," J. OF MEASUREMENT & AUTOMATIC CONTROL SOCIETY, in print.

8. Ishikawa, "Method To Detect the Center and Sum of Output Distribution from Sensors in Matrix," Ibid., Vol 19, No 5, 1983, pp 381-386.

9. Ibid., "Passive Sensor System by Parallel Processing," submitted.

10. H.M. Raibert and J.E. Tanner, "Design and Implementation of a VLSI Tactile Sensing Computer," INT. J. ROBOTICS RES., Vol 1, No 3, 1982, pp 3-18.

11. T.C. Henderson and E. Shilcrat, "Logical Sensor System," J. ROBOTIC SYST., Vol 1, No 2, 1984, pp 169-193.

12. Ishikawa, "Local Pattern Processing LSI With Parallel Processing and Its Application to Tactile Sensors," J. OF MEASUREMENT & AUTOMATIC CONTROL SOCIETY, Vol 24, No 3, 1988, pp 228-235.

13. Ibid., "Tactile Sensor With Parallel Processing Functions," SHINGAKU-GIHO, ICD88-3, 1988.

14. L.D. Harmon, "Automatic Tactile Sensing," INT. J. ROBOTICS RES., Vol 1, No 2, 1982, pp 3-32.

15. Ishikawa, "Tactile System," COMPUTER ROLE, No 21, 1988, pp 59-66.

16. Maeda, "Graphics Processing Machine," INFORMATION PROCESSING, Vol 28, No 1, 1987, pp 19-26.

17. Ishikawa and Maeda, "Design of Transputer Board with MULTIBUS Spec. and Its Application to FA System," 32-bit Microprocessor Application-Development-Evaluation, Nikkei McGraw-Hill, 1988.

18. Ishikawa, "Sensor Fusion System--Integrated Mechanism of Tactile Information, J. OF JAPAN ROBOTICS SOCIETY, Vol 6, No 3, 1988, in press.

19. Ishikawa, et al., "Development of Sensor Processing Language by Concept of Logical Sensors," Measurement & Automatic Control Society, the 24th Conference preprint, 1985, pp 49-50.

20. Amari, "Neuro Computation--Aiming at Parallel Learning Information Processing," MEASUREMENT & CONTROL, Vol 27, No 3, 1988, pp 255-263.

20149/9365 - END -

10

SUBSCRIPTION/PROCUREMENT INFORMATION

The FBIS DAILY REPORT contains current news and information and is published Monday through Friday in 8 volumes: China, East Europe, Soviet Union, East Asia, Near East & South Asia, Africa (Sub-Sahara), Latin America, and West Europe. Supplements to the DAILY REPORTs may also be available periodically and will be distributed to regular DAILY REPORT subscribers. JPRS publications generally contain less time-sensitive information and are published periodically. Current JPRS publications are listed in *Government Reports Announcements* issued semi-monthly by the National Technical Information Service (NTIS), 5285 Port Royal Road, Springfield, Virginia 22161 and the *Monthly Catalog of U.S. Government Publications* issued by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

U.S. Government offices may obtain subscriptions to the DAILY REPORTs or JPRS publications (hardcovers or microfiche) at no charge through their sponsoring organizations. DOD consumers are required to submit requests through appropriate command validation channels to DIA, RTS-2C, Washington, D.C. 20301. (Telephone: (202) 373-3771, Autovon: 243-3771.) For additional information or assistance, call FBIS, (703) 527-2368, or write to P.O. Box 2604, Washington, D.C. 20013.

The public may subscribe to either hardcover or microfiche versions of the DAILY REPORTs and JPRS publications through NTIS at the above address or by calling (703) 487-4630. Subscription rates will be provided by NTIS upon request. Subscriptions are available outside the United States from NTIS or appointed foreign dealers. Back issues or single copies of the DAILY REPORTs and JPRS publications are not available. New subscribers should expect a 30-day delay in receipt of the first issue.

Both the DAILY REPORTs and the JPRS publications are on file for public reference at the Library of Congress and at many Federal Depository Libraries. Reference copies may also be seen at many public and university libraries throughout the United States.