大阪公立大学
Osaka Metropolitan University
学術情報リポジトリ

A Continuous System Simulation Method Using Multimicroprocessor System TCMS

メタデータ	言語: English 出版者: 公開日: 2010-04-06 キーワード (Ja): キーワード (En): 作成者: Kosako, Hideo, Mihara, Jun-ichi, Kojima, Yoshiaki メールアドレス: 所属:
URL	https://doi.org/10.24729/00008555

A Continuous System Simulation Method Using Multimicroprocessor System TCMS

By Hideo KOSAKO*, Jun-ichi MIHARA*, and Yoshiaki KOJIMA*

(Received Nov. 15, 1984)

This paper describes a new simulation method for continuous systems and tests it by experiments with a compact multimicroprocessor system TCMS we have recently developed. In the proposed method operational tasks are allocated automatically to processors during the multioperation without using any task scheduling technique.

1. Introduction

On the digital simulation of continuous systems, most of recent studies tend to improve the operation speed and the cost performance by using a large number of processors. These processors constitute all together a simulator for continuous systems. There are a number of such simulators in active use. HOSS[1] consists of thirty-four high-performance mini-computers; FKCSS[2] consists of sixty-four microprocessors; IDDS-1[3] is a small scale multimicroprocessor system (using Z80A) designed to examine an optimum task scheduling, etc.

There is a fixed task allocation method employed in these simulators. The method allocates operational tasks in advance to a processor for the implementation of the continuous system simulation. The disadvantages of this method are:

(1) There is a reduction in the rate of the utilization of processors.

(2) It is impossible to carry out the multioperation if the number of the tasks involved exceeds that of the processors.

To circumvent these disadvantages, a task scheduling technique[3] has been used. In this case, however, it is not easy to write a program which gives an optimum scheduling.

For this reason a new method is proposed in this paper, which requires no scheduling procedure for task allocation. In the present system it is assumed that a packet is a simulation language which can simulate approximately the function of an analogue computing element. Then, a simulation program, which is named "packet program", is a set of such packets and is stored in the common memory of the multiprocessor system. The processors thus hold the packet program in common.

Each processor fetches a packet from the packet program and interprets it as a single instruction to execute a specified task independently, and can fetch another packet immediately after the completion of the task.

Thus, in the method described here, in which packets are used as media of allocation, the task allocation is automatically carried out to processors. On the contrary, the scheduling in the fixed task allocation method is regarded as manual allocation of tasks to processors.

* Department of Electronics, College of Engineering.

2. Parallel operations for the continuous system

The simulation method to be described will be applicable only to a system of K ordinary differential equations

$$\left.\begin{aligned} \frac{dy_j}{dx} &= f_j(y_1, y_2, \cdots\cdots, y_K) \\ j &= 1, 2, \cdots\cdots, K \end{aligned}\right\} \qquad (1)$$

where f_j may be non-linear operations such as multiplication, division, and so on.

This section outlines the configuration of the multimicroprocessor system which we constructed and named TCMS, and describes a new representation of Eq. (1) adapted and computation procedures of performing a packet program.

2.1 Multimicroprocessor system, TCMS[4)]

A block diagram of the TCMS is shown in Fig. 1(a). The TCMS is designed to have a cluster of up to eight processor units (PUs) which have their own local memories (LMs) and work independently. All of these processor units are connected mutually by time-shared buses via the common memory (CM).

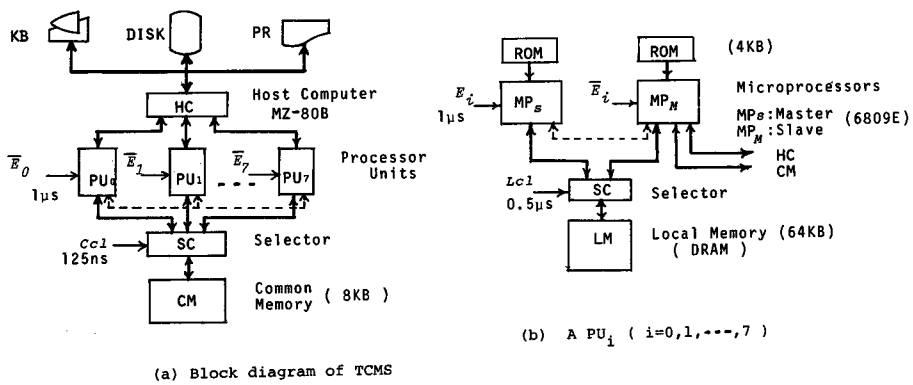

Fig. 1 Multi-microprocessor system, TCMS

Each processor unit has two 6809E microprocessors, one of which is called "master microprocessor" MP_M and the other "slave microprocessor" MP_S. Let T denote the memory cycle time for the storage device of the 6809E. Then the two microprocessors in a PU can obtain words through a selector by turns from the local memory (LM) at intervals of $T/2$.

The CM and a host-computer (HC) are connected to the master microprocessors (MP_M s) in the PUs as shown in Fig. 1 (b). Clock $\overline{E}_i$ ($i \in I, I = \{0, 1, \cdots\cdots, 7\}$) is fed to the MP_M of PU_i ($i \in I$). The timing chart for $\overline{E}_i$ is given in Fig. 2. Each clock has a delay of $T/8$ from the previous one.

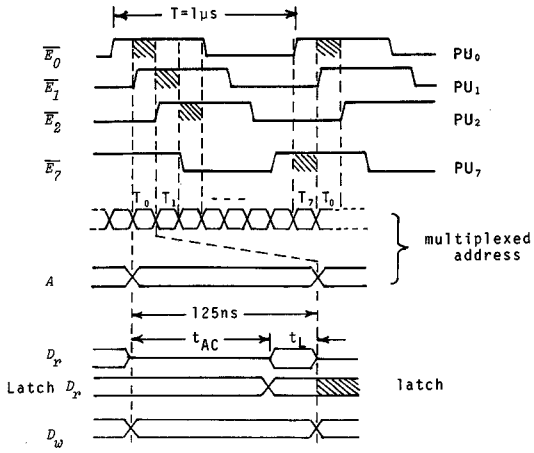

Fig. 2 Access timing of each PU to CM.

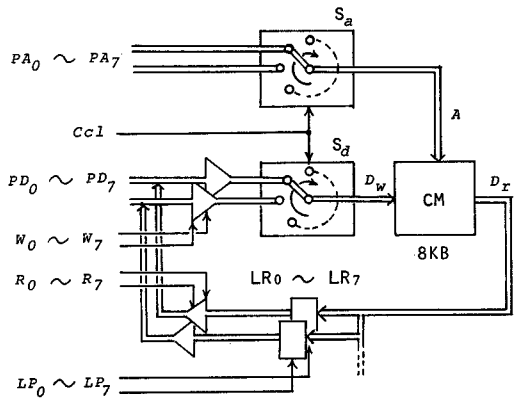

Fig. 3 Selector switches in the TCMS

Two multiplexers (one of which is an address bus multiplexer and the other a data bus multiplexer) and latch registers are used to connect the eight PUs to the CM, as shown in Fig. 3.

Both multiplexers are switched to combine the buses from all the PUs in turn to the CM at intervals of $T/8$.

Let τ_a be the access time of the CM. If

$$\tau_a \leqq T/8 \tag{2}$$

then a $T/8$-interval time-slot is assigned to each PU for access to the CM.

In Fig. 2, the symbols T_0, T_1, ·····, T_7 are time-slots assigned to the processor units PU_0, PU_1, ·····, PU_7, respectively. The 6809E microprocessors of the PUs are known to have 1 μs memory cycle time ($T = 1\ \mu s$) and hence the time-slot is 125ns in width in the present system.

The symbol $\mathrm{LR}_i(i \in I)$ in Fig. 3 denotes the buffer register in which the data D_r read out of the CM is placed. The symbol $LP_i(i \in I)$ denotes the pulse which is used to send the D_r to the buffer register LR_i. By the read instruction R_i, the data in the LR_i is transferred to the PU_i through the data bus.

Let D and A denote a data and an address in the CM, respectively. If a PU_i and a $\mathrm{PU}_k\{k = mod_8(i+1)\}$ perform, respectively, parallel operations 'Write D to address A' and 'Read address A, then the data D is moved from the PU_i to the PU_k at the transfer rate of 125 ns.

2.2 A program of continuous systems - - - - packet program

A packet as a simulation language is assumed to be able to simulate the function of an analogue computing element in the analogue program required for solving Eq. (1).

Let P_j denote the packet corresponding to the analogue computing element $j(j \in J,\ K \leqq M,\ J = \{1, 2, \cdots\cdots, M\})$, and be representated in the form

$$P_j = \Phi_I(\boldsymbol{a}_j, \boldsymbol{Y}_j, \boldsymbol{F}_j) \text{ or } P_j = \Phi_I^{\sim}(\boldsymbol{a}_j, \boldsymbol{Y}_j) \tag{3}$$

Here, Φ_I denotes a summing integrator and $\Phi_I^{\sim}$ indicates an operator except Φ_I, for example, adder (Φ_A) or multiplier (Φ_M).

In Eq. (3),

$$\left.\begin{aligned} &\boldsymbol{a}_j = (a_{jk}, a_{jl}, \cdots\cdots) \\ &\boldsymbol{Y}_j = (Y_j^1, \boldsymbol{Y}_j^0),\ \boldsymbol{Y}_j^0 = (Y_j^0, Y_k^0, Y_l^0, \cdots\cdots) \\ &\qquad \text{for } j, k, l \in J,\ J = \{1, 2, \cdots\cdots, M\} \\ &\boldsymbol{F}_j = (F_j^0, F_j^{-1}, F_j^{-2}, \cdots\cdots) \\ &\qquad \text{for } j \leqq K \end{aligned}\right\} \tag{4}$$

where, for instance, a_{jk} is the coefficient between the output of analogue computing element k and the input of element j; Y_j^1, Y_k^0, Y_l^0, F_j^0, F_j^{-1}, $F_j^{-2}\cdots$ are the address numbers in the CM, which will be described in detail later.

A set of packets $\{P_j | j \in J,\ J = \{1, 2, \cdots M\}\}$, i.e, a packet program, is stored in the CM with the data that is the contents in addresses $\boldsymbol{Y}_j(j = 1, 2, \cdots M)$ and $\boldsymbol{F}_j(j = 1, 2, \cdots K)$. Each packet in the program is read out of the CM by a PU, and it operates as a single instruction to the PU.

2.3 Principle of numerical integration for multiprocessing of packets

Consider Eq. (1) to be the homogeneous linear equations

$$\left.\begin{aligned} &\frac{dy_j}{dx} = f_j, \quad f_j = \sum_{k=1}^{K} a_{jk} y_k \\ &j = 1, 2, \cdots\cdots\cdots K \end{aligned}\right\} \qquad (5)$$

with constant coefficients $a_{jk}'s$. The program for solving Eq. (5) can be expressed as a set of integration packets mentioned in section 2.2,

$$\left.\begin{aligned} &P_j = \Phi_I(\boldsymbol{a}_j, \boldsymbol{Y}_j, \boldsymbol{F}_j) \\ &j = 1, 2, \cdots\cdots\cdots, K \end{aligned}\right\} \qquad (6)$$

where Φ_I is the operator which instructs a PU to perform a numerical integration.

The integration by operator Φ_I is carried out for a fixed value $h > 0$, sufficiently small increment of the independent variable x.

Let the initial condition be $y_j(x_o) = y_{jo}$ where $j = 1, 2, \cdots\cdots, K$ and set $x_n = x_o + nh$, $n = 0, 1, 2, \cdots$. Let y_{jn} and y'_{jn} denote the numbers at the nth step of approximation to calculate $y_j(x_n)$ and $f_j(x_n, y_{1n}, y_{2n}, \cdots, y_{Kn})$, respectively. Then, y_{jn} and y'_{jn} can be expressed as follws:

$$\left.\begin{aligned} &y_{in} \approx y_i(x_n) \\ &y'_{jn} = f_{jn} = f_j(x_n, y_{1n}, y_{2n}, \cdots\cdots\cdots, y_{Kn}) \approx y'_j(x_n) \\ &n = 0, 1, 2, \cdots\cdots\cdots \end{aligned}\right\} \qquad (7)$$

If the round-off error is negligible, the difference $y_{jn} - y_j(x_n)$ includes only the truncation error at the nth step, where $y_j(x_n)$ is the exact solution.

Integration formulas may be divided into two classes, 'open' and 'close' types: the former are effective to obtain predictors and the latter useful for correctors. The 'open' integration formulas are used to get solutions explicitly. In the 'open' type formulas, $y_{j(n+1)}$, the approximate solution at the (n+1) th step, is expressed as a linear combination of values of f_{ji} for $0 \leqq i \leqq n$.

Thus, the operation of packet program uses the formulas of this type to solve differential equations numerically. Some examples of the 'open' integration formulas are given below.

Let $y_{j(n+1)}$ be expressed in the form

$$\left.\begin{aligned} y_{j(n+1)} &= y_{jn} + h \cdot g_{jn} \\ j &= 1, 2, \cdots\cdots\cdots, K \\ n &= 0, 1, 2, \cdots\cdots\cdots \end{aligned}\right\} \qquad (8)$$

Then the following forms are used as g_{jn}.

$$g_{jn} = f_{jn} \qquad (9-a)$$

$$g_{jn} = (3 f_{jn} - f_{j(n-1)})/2 \qquad (9-b)$$

$$g_{jn} = (23 f_{jn} - 16 f_{j(n-1)} + 5 f_{j(n-2)})/12 \qquad (9-c)$$

$$g_{jn} = (55 f_{jn} - 59 f_{j(n-1)} + 37 f_{j(n-2)} - 9 f_{j(n-3)})/24 \qquad (9-d)$$

where, according to Eqs. (5) and (7),

$$f_{jn} = \sum_{k=1}^{K} a_{jk} y_{kn} \qquad (10)$$

Equation (9-a) shows the Euler rule. Equations (9-b), (90c) and (9-d) are known as the 2nd-order, 3rd-order, and 4th-order Adams-Bashforth rules, respectively.

Now, the purpose of the operator Φ_I in the packet P_j is to evaluate $y_{j(n+1)}$ and f_{jn} in terms of Eqs. (8), (9) and (10). In the case of (9-d), for example, the (n+1) th step of the integration requires the knowledge of the past values of $y_{1n}, y_{2n}, \cdots,, y_{jn}, \cdots, y_{Kn}$ and $f_{j(n-1)}, f_{j(n-2)}, f_{j(n-3)}$, which are assumed to be stored in addresses $Y_1^0, Y_2^0, \cdots, Y_j^0, \cdots, Y_K^0$ and $F_j^{-1}, F_j^{-2}, F_j^{-3}$ respectively. The resulting values $y_{j(n+1)}$ and f_{jn} are stored in addresses Y_j^1 and F_j^0, respectively.

After $y_{1(n+1)}, y_{2(n+1)}, \cdots, y_{K(n+1)}$ and $f_{1n}, f_{2n}, \cdots, f_{Kn}$ are found, the data are transferred to prepare for the next step in the computation as follows:

$$(Y_j^1) \rightarrow Y_j^0, \quad (F_j^{-2}) \rightarrow F_j^{-3}, \quad (F_j^{-1}) \rightarrow F_j^{-2}$$

and $(F_j^0) \rightarrow F_j^{-1}$ for $j = 1, 2, \cdots\cdots\cdots K$

where the parentheses are used to indicate the contents in the memory addresses, namely, the notation $(Y_j^1) \rightarrow Y_j^0$ means the contents of address Y_j^1 to be stored into address Y_j^0.

The choice of which 'open' integration formula rules are used can be attained merely by describing it in the packet. For example, if the 3rd-order Adams-Bashforth rule (9-c) is chosen, then the description in the packet P_j is as follows:

$$\left.\begin{aligned} &P_j = \Phi_{I3}(\boldsymbol{a}_j,\ \boldsymbol{Y}_j,\ \boldsymbol{F}_j) \\ &\boldsymbol{a}_j = (a_{jk},\ a_{jl},\ \cdots\cdots\cdots) \\ &\boldsymbol{Y}_j = (Y_j^1,\ \boldsymbol{Y}_j^0),\ \boldsymbol{Y}_j^0 = (Y_j^0\ ,\ Y_k^0,\ Y_l^0,\ \cdots\cdots\cdots) \\ &\boldsymbol{F}_j = (F_j^0,\ F_j^{-1},\ F_j^{-2}) \end{aligned}\right\} \tag{11}$$

where, for instance, Y_k^0 is the shared memory address where y_{kn} is stored. The value y_{kn} is used to compute $y_{j(n+1)}$.

Note that to compute $y_{j(n+1)}$ requires the knowledge of y_{kn}, $y_{k(n-1)}$, $y_{k(n-2)}$ or $y_{k(n-3)}(k=1,\ 2,\ \cdots K)$ but at the start$(n=0)$only the value y_{ko} are known. The points at which the starting values are computed can be successive samples of values $x_i > x_o$. For the ith-order Adams-Bashforth rule these points are x_1, x_2, $\cdots$, and x_{i-1}.

The first few starting values must be computed with an accuracy at least as high as that of the numerical integration procedure to be used. There is no formula for choosing the value of h with which to start the integration. If the Euler rule is used, then only one initial condition is required to start computation. For a given integer $N' \geqq 2$, let $h' = h/N'$, and integer $n' < (i-1)N'$. Then the Euler rule for the computation at the $(n'+1)$ th step is expressed as follws:

$$\left.\begin{aligned} &\tilde{y}_{j(n'+1)} = \tilde{y}_{jn'} + h'\tilde{f}_{jn'} \\ &\quad j = 1,\ 2,\ \cdots\cdots\cdots,\ K;\ \ n' = 0,\ 1,\ \cdots\cdots\cdots,\ (i-1)N'-1 \end{aligned}\right\} \tag{12}$$

where

$$\tilde{y}_{jn'} \approx y_{j(x_0+h'n')}$$

$$\tilde{f}_{jn'} \approx f_j(x_0 + h'n',\ \tilde{y}_{1n'},\ \tilde{y}_{2n'},\ \cdots\cdots\cdots,\ \tilde{y}_{Kn'})$$

moreover

$$\tilde{y}_{j(kN')} = y_{jk}$$

$$\tilde{f}_{j(kN')} = f_{jk} \ \text{ for } k = 0,\ 1,\ \cdots\cdots\cdots,\ i-1.$$

2.4 Dynamic processing procedures of packet programs

The chief function of a PU is to perform the instructions contained in the packet which is sent from the CM to the LM of the PU. Now, the computation procedure

for performing a packet program with the multiprocessor system will be described. Consider, as an example, the block diagram of an integration packet P_j shown in Fig. 4. The packet P_j is expressed as follows:

$$P_j = \Phi_{I2}(\boldsymbol{a}_j, \boldsymbol{Y}_j, \boldsymbol{F}_j) \tag{13}$$

where

$$\boldsymbol{a}_j = (a_{jk}, a_{jl})$$

$$\boldsymbol{Y}_j = (Y_j^1, Y_j^0, Y_k^0, Y_l^0)$$

$$\boldsymbol{F}_j = (F_j^0, F_j^{-1})$$

Note that the 2nd-order Adams-Bashforth rule is chosen as a integration formula.

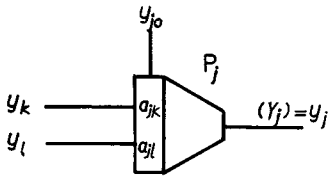

Fig. 4 The block diagram of an integration packet P_j

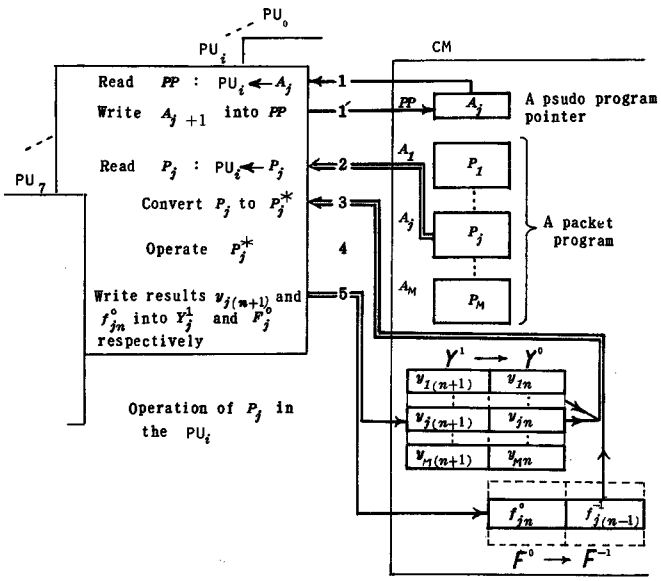

Fig. 5 The procedure of processing of a P_j ($j \in J$)

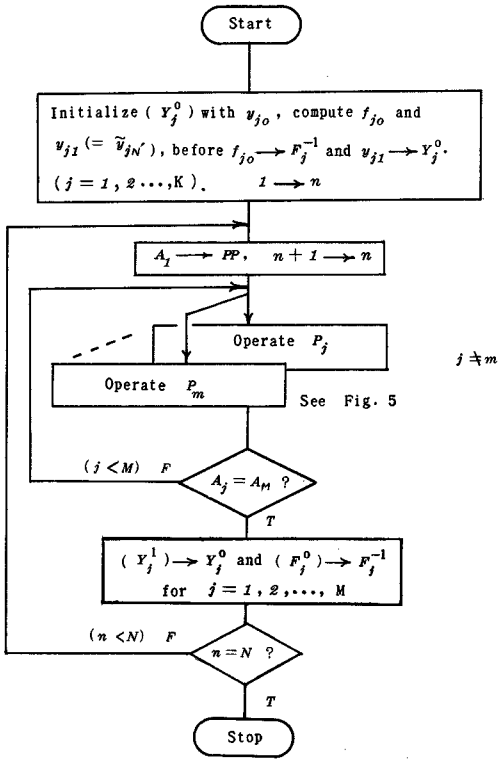

Fig. 6 Iterative operation of program
N : A number of iterations required

Figure 5 shows the procedure for processing the packet P_j after a PU_i ($i \in I$, $I = \{0, 1, \cdots, 7\}$) fetches the P_j from the CM, assuming that the packet P_j is stored in the successive addresses beginning at A_j.

Step 1 and 1′: At the starting point of taking a packet from the CM, a PU_i reads the content A_j of the address pp which indicates the psudo program pointer. Then, the PU_i replaces the contents of the pp, namely A_j with A_{j+1}.

Step 2: The PU_i takes a packet P_j stored in the successive addresses beginning at A_j.

Step 3: Then, the PU_i converts the P_j into the following object packet P_j^*:

$$P_j^* = \Phi_{12}(\boldsymbol{a}_j, \boldsymbol{y}_j, \boldsymbol{f}_j)$$

where $\boldsymbol{y}_j$ and $\boldsymbol{f}_j$ are the contents of addresses $\boldsymbol{Y}_j$ and $\boldsymbol{F}_j$, respectively.

Step 4: The operation program expressed by Φ_{I2} is performed. (Then $y_{j(n+1)}$ and f_{jn} are obtained using Eqs. (8), (9-b) and (10).)

Step 5: $y_{j(n+1)}$ and f_{jn} are stored at addresses Y_j^1 and F_j^0, respectively.

Described above is the way how a single PU processes a single packet for a fixed value x in the computation. If more than one PU are used, packets are processed simultaneously. By repeating the above procedures for a given packet program $\{P_j | j = 1, 2, \cdots, M\}$, the simulation is accomplished as shown in Fig. 6

We call the above technique a shared-packet-program method for the continuous system simulations.

3. Examples of the continuous system simulation using the TCMS

This section describes simple examples of the continuous system simulation using the TCMS. The specifications for the system used are as follows:

(1) Classification of operation packet: Integrator, adder and multiplier

(2) Maximum number of packets: 128

(3) Number system: Floating-point number system (4 byte-length)

(4) Numerical integration formulas in use: Euler rule and the 2nd-order Adams-Bashforth rule (which will be called A-B rule for short)

(5) Number of processor units: 4 or 8

The first example is to check on the truncation error of the formulas when a single processor unit is used. The tests are performed for simple programs which give solutions $e^x (x > 0)$, $\sin x$, $\cos x$, and so on. One of these results is shown in Table 1. The values in Table 1 are the truncation errors for e^x obtained by the packet program shown in Fig. 7. It can be seen, from Table 1, that the 2nd-order A-B rule is about thirty times as fast as the Euler rule in the computation speed of obtaining the same order of accuracy.

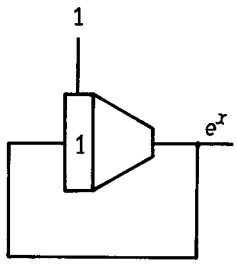

Fig. 7 A program for a test of truncation errors

Table 1 Truncation errors of a function e^x solved by the packet program (Fig. 7).
Op. time denotes a real time required to the operation by $x = 10$ for a starting value $x = 0$.

By Euler rule

x	$h = 2^{-4}$	$h = 2^{-6}$	$h = 2^{-8}$	$h = 2^{-10}$	$h = 2^{-12}$
2	5.82E-2	1.53E-2	3.89E-3	9.76E-4	2.42E-4
4	1.13E-1	3.05E-2	7.76E-3	1.95E-3	4.87E-4
6	1.65E-1	4.53E-2	1.16E-2	2.92E-3	7.32E-4
8	2.13E-1	6.00E-2	1.55E-2	3.89E-3	9.75E-4
10	2.59E-1	7.44E-2	1.93E-2	4.87E-3	1.22E-3
Op. time	0.6(sec)	2(sec)	6(sec)	23(sec)	92(sec)

By Adams-Bashforth rule

x	$h = 2^{-4}$	$h = 2^{-6}$	$h = 2^{-8}$	$h = 2^{-10}$	$h = 2^{-12}$
2	4.94E-3	3.22E-4	2.04E-5	3.67E-6	1.73E-6
4	8.05E-3	5.23E-4	3.22E-5	2.40E-6	1.08E-6
6	1.11E-2	7.24E-4	4.51E-5	2.46E-6	1.12E-6
8	1.42E-2	9.25E-4	5.66E-5	3.31E-7	-1.72E-7
10	1.73E-2	1.13E-3	6.88E-5	-5.54E-7	3.61E-8
Op. time	0.8(sec)	3(sec)	10(sec)	39(sec)	162(sec)

The second example is to solve a Van der Pol equation

$$\frac{d^2y}{dt^2} - \epsilon\,(l-y^2)\frac{dy}{dt} + y = 0 \tag{14}$$

with the initial conditions $\dot{y}(0) = 0$ and $y(0) = 2^{-4}$. Let $y_1 = dy/dt$ and $y_2 = y$ in Eq. (14). Then Eq. (14) can be written as

$$\left.\begin{aligned} \frac{dy_1}{dt} &= \epsilon\,(y_1 - y_3) - y_2 = f_1 \\ \frac{dy_2}{dt} &= y_1 = f_2 \\ y_3 &= y_1 \cdot y_2^2 \end{aligned}\right\} \tag{15}$$

With Eqs. (8), (9-b) and (10), the solutions y_1 and y_2 of Eq. (15) are fround approximately as successive samples of values y_{1n} and y_{2n} $(n = 0, 1, 2, \cdots)$, respectively. The initial conditions are $y_{10} = \dot{y}(0) = 0$ and $y_{20} = y(0) = 2^{-4}$. The program solving Eq. (15) is shown in Fig. 8. This program can be expressed in the packet program

$$P_1\text{: } \Phi_{I2}(a_{11}, a_{12}, a_{13}, Y_1^1, Y_1^0, Y_1^0, Y_2^0, Y_3^0, F_1^0, F_1^{-1})$$

$$P_2\text{: } \Phi_{I2}(a_{21}, Y_2^1, Y_2^0, Y_1^0, F_2^0, F_2^{-1})$$

$$P_3\text{: } \Phi_M(a_{31}, a_{32}, a_{32}, Y_3^1, Y_3^0, Y_1^0, Y_2^0, Y_2^0)$$

where $a_{11}=-a_{13}=\epsilon$, $a_{21}=a_{31}=a_{32}=1$ and $a_{12}=-1$.

By using Eq. (12), the starting values for the 2nd-order A-B rule can be obtained as follows:

$$\left.\begin{aligned}
&\tilde{y}_{j(n'+1)}=\tilde{y}_{jn'}+h'\tilde{f}_{jn'}\ (j=1,\ 2)\\
&\tilde{f}_{1n'}=\epsilon\ (\tilde{y}_{1n'}-\tilde{y}_{3n'})-\tilde{y}_{2n'}\\
&\tilde{f}_{2n'}=\tilde{y}_{1n'}\\
&\tilde{y}_{3(n+1)}=\tilde{y}_{1n'}\cdot\tilde{y}_{2n'}^2\\
&\qquad n'=0,\ 1,\cdots\cdots\cdots N'-1;\ N'=h/h'
\end{aligned}\right\}\tag{16}$$

where $\tilde{y}_{jN'}=y_{j1}$, $\tilde{y}_{j0}=y_{j0}\ (j=1,\ 2,\ 3)$

and $\tilde{f}_{jN'}=f_{j1}$, $\tilde{f}_{j0}=f_{j0}\ (j=1,\ 2)$

The values $\tilde{y}_{1n'}$ and $\tilde{y}_{j(n'+1)}$ are stored into addresses Y_j^0 and Y_j^1, respectively, and the values $\tilde{f}_{j0}$ and $\tilde{f}_{jN'}$ are stored into addresses F_j^{-1} and F_j^0, respectively. The solution starts to find y_{j2} using the 2nd-order A-B Rule after the values of y_{j1} and $f_{j1}(j=1,\ 2,\ 3)$ are computed with Eq. (16) for the initial conditions $y_{10}=0$ and $y_{20}=2^{-4}$.

The waveforms in Fig. 9 are obtained by using the packet program for Eq. (14) in the case of $\epsilon=1$ and $h=2^{-6}$, $h'=2^{-10}(N'=2^4)$. The waveform y_{AB} is the result computed by the 2nd-order A-B rule and the waveform δ shows the difference $\delta=y_E-y_{AB}$ where y_E denotes the result by the Euler rule.

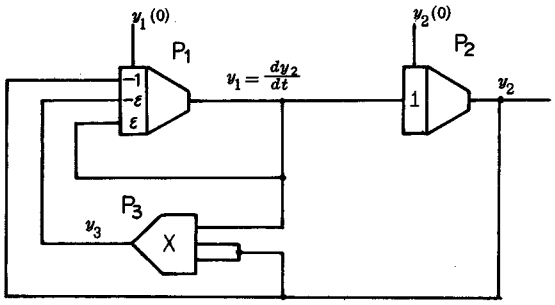

Fig. 8 A program to solve Van der Pol equation

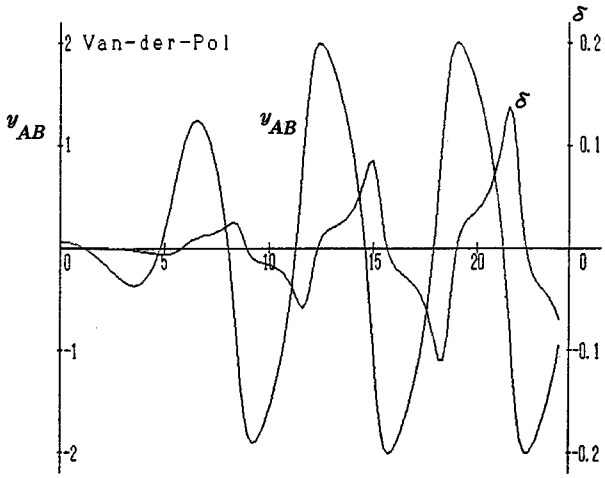

Fig. 9 A solution of Eq. (14), y_{AB}, and difference values $\delta = y_E - y_{AB}$

The last example is to examine the function of the TCMS on the parallel processing. Let m and M denote the number of processors and packets, respectively. In general the processing time of a packet program may be considered, in the case of $M \gg m$, to be proportional to M and inversely proportional to m. Let $t(M_1, m_1)$ and $t(M_2, m_2)$ denote, respectively, the processing times for $M = M_1$, $m = m_1$ and $M = M_2$, $m = m_2$. Then, the following relationship will hold:

$$t\,(M_1,\, m_1) = \frac{M_1 m_2}{M_2 m_1}\, t\,(M_2,\, m_2) \tag{17}$$

Figure 10 shows the test program used. The program consists of M packets with M-2 dummy packets among them, where M takes a value of the interval $2 \leqq M \leqq 128$.

The results of executing the test program for $M = 2^k$, $k = 1, 2, \cdots, 7$ are shown in Table 2, From Table 2, for instance, t(8,4) = 4.4 msec and t(64,8) = 16.8 msec. Hence t(64,8) $\approx$ 4t(8,4), which, as expected, satisfies Eq. (17) approximately.

It is clear from the above that multiprocessor system is of high level in the throughput for parallel operation of packets when $M \gg$ m and M/m is an integer.

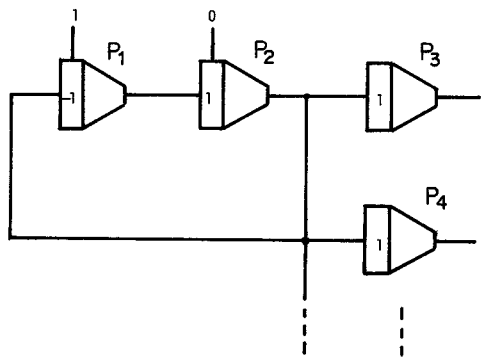

Fig. 10 A test program for examining the function of the parallel processing

Table 2 The time required to multi-operate M packets with m processor-units for a stepwise h.

M	t (M, m) msec	
	t(M,4)	t(M,8)
2	2.4	2.4
4	2.6	2.4
8	4.4	4.2
16	8.0	5.8
32	15.2	9.4
64	29.6	16.8
128	58.6	31.6

4. Conclusion

In this paper we described a new method for simulating continuous systems by using the multiprocessor system TCMS.

The advantages of this method are:

(1) Since each analogue computing element can be expressed by packet form, it is easy to write a simulation program by using the packets.

(2) Tasks are allocated to processors automatically by means of a shared-packet-program method. In this case it is not necessary to use the task scheduling technique, which is invented for the purpose of improving the throughput in the fixed task allocation method.

(3) The more the number of packets in a given packet program exceeds that of processors, the higher the multiprocessor system is with regard to the throughput.

Thus we come to a conclusion that the simulation method presented here could be applied to the following general-purpose multimicroprocessor systems: (1) Every processor unit is linked to a shared memory (or common memory) in which a packet program is stored. (2) Each processor unit has the private memory (local memory) with a capacity for storing a packet interpreter and an operational task library.

References

1) S. Koyama, K. Makino, N. Miki, J. Simulation, 1, 42 (1981).
2) T. Nakagawa, J. SICE, 21, 477 (1982).
3) H. Kasahara, S. Narita, J. Simulation, 2, 142 (1983).
4) H. Kosako, Y. Goto, H. Kisaku, Y. Kojima, J. Simulation, 3, 113 (1984).