ELSEVIER

Simulation Practice and Theory 5 (1997) 137–151

SIMULATION PRACTICE AND THEORY

A methodology for digital real time simulation of dynamic systems using modern DSPs

Krešimir Ćosić [a,*], Ivica Kopriva [a,1], Tomislav Šikić [b,2]

[a] *Faculty of Electrical Engineering, Vukovarska 39, 10000 Zagreb, Croatia*

[b] *Department of Mathematics, University of Zagreb, Bijenička 30, 10000 Zagreb, Croatia*

Received 1 December 1994; revised 5 March 1996

Abstract

The speedup factor in real time simulation of dynamic systems using multiprocessor resources depends on: the architecture of the multiprocessor system, type of interconnection between parallel processors, numerical methods and techniques used for discretization and task assignment and scheduling policy. The minimization of the number of processors needed for real time simulation requires the minimization of processors times for interprocessor communications and efficient scheduling policy. Therefore, this article presents a methodology for the real time simulation of dynamic systems including a new pre-emptive static assignment and scheduling policy. The advantages of applying digital signal processor with parallel architecture, for example TMS320C40, in real time simulation have been described. Some important issues in real time architectures necessary for efficient multiprocessor real time simulations, such as multiple I/O channels, concurrent I/O and CPU processing, direct high speed interprocessor communications, fast context switching, multiple busses, multiple memories, and powerful arithmetic units are inherent to this processor. These features minimize interprocessor communication time and maximize sustained CPU performance.

Keywords: Real time simulations; Parallel processing; Pre-emptive static scheduling policy; Hard deadline periodic tasks; Digital signal processors; Asynchronous parallel interprocessor communication

* Corresponding author.

[1] Email: ikopriva@jagor.srce.hr.

[2] Email: sikic@math.hr.

PII S0928-4869(96)00003-1

1. Introduction

The current importance and interest for tactical training systems, distributed interactive simulation, virtual reality, man-in-the-loop training and hardware in the loop simulation require an efficient methodology for real time simulation using multiprocessor resources. These training or testing scenarios include a number of different moving objects, dynamic systems or subsystems, which are characterized by the appropriate mathematical model. Efficient multiprocessor real time simulation of such complex mathematical models requires their decomposition into clearly defined subsystems preserving physical and logical structure of the original problem. Such physical and logical decomposition represents a systematic and effective technique for discovering and utilizing concurrence in parallel real time simulation of complex systems which have to be distributed among a number of processors. Such structural decomposition at the same time represents a basis for partitioning of the starting simulation problem into executable module/task units and their assignment and scheduling. The allocation of modules/tasks, which preserves the physical structure of the original model, minimizes interprocessor communication and makes testing and debugging of real time simulation problem much easier.

The methodology by which a given simulation problem is partitioned, discretized, distributed, and activated determines the efficiency of multiprocessor implementation, i.e., the required number of microprocessors and percentage of their utilization.

Minimization of the number of microprocessors needed for real-time simulation requires optimal utilization of each microprocessor as well as the minimization of their mutual intercommunications. The basic criterion for the task assignment and scheduling policy, the module/task deadlines and module/task workload parameters have been introduced. The module/task deadline is determined by the discretization period and presents the hard timing constraint imposed on the execution of each task while the module/task workload presents the required intensity of computations for module/task processing during the module/task deadline period. The new pre-emptive static algorithm for scheduling hard deadline periodic tasks is introduced. This algorithm, called clustering scheduling policy, approaches 100% of utilization factor, like dynamic deadline driven scheduling policy [3,9,10], but is at the same time easier for software implementation as other static scheduling algorithms [3,10,14]. The starting global workload for multiprocessor implementation strongly depends on the applied numerical methods and techniques for discretization of the original continuous problem. This workload, using suitable numerical methods and techniques, can be minimized so that the arithmetic complexity of discretized model can be significantly reduced using highly efficient numerical methods, see [1,2,4,5,7,11].

2. Methodology for a multiprocessor real-time simulation

Real time simulation of dynamic systems is essentially signal processing application composed of tasks with hard deadline periodic requests, and therefore, requires deterministic task executions, i.e., deterministic response times which are determined by task deadlines. Any breakthrough of these hard real time constraints may cause

data losses and may throw the real time simulation into unknown states. Therefore, in real time simulation, hard real time constraints are the inherent part of the problem definition [1,3,9,10,13,14]. Failure to observe all timing constraints imposed on the modules/tasks deadlines leads to severe consequences and "may result in economic, human and ecological catastrophes" [15]. Therefore, all timing constraints must be explicitly and clearly specified by modules/tasks computation times and corresponding deadlines. In that sense, each simulation task is characterized by a pair T and C, i.e., $\tau_i = (C,T)$, where T denotes discretization period, i.e., deadline period and C is task run-time. Real-time simulation requires that the latest possible completion time of each module/task is less than its deadline, i.e., $C < T$. "The scheduling problem is to specify an order in which the requests of a set of tasks are to be executed and the processor to be used, with the goal of meeting all the deadlines with a minimum number of processors" [9], while "a scheduling algorithm is a set of rules that determine the task to be executed at a particular moment" [10]. The pre-run-time assignment and scheduling policy is applied to the task set in order to ensure the temporal correctness and optimal utilization of hardware resources required for real time implementation. In such timing organization there is no need for the task message synchronization since all tasks share the module/task variables at the precise moment of real time. To describe a more systematic approach to the real-time simulation using multiprocessor resources, the following definitions are introduced:

Definition 1. The *design objective function* requires real-time simulation of a given problem with prescribed accuracy requirements using a minimal number of microprocessors. The objective function is simultaneously looking for:

- the discrete model with minimal workload characterization,
- the optimal utilization of each microprocessor separately,
- the minimization of microprocessor intercommunications.

Definition 2. *Module/task* represents a logical and consistent portion of a global simulation problem. The module/task corresponds to some physical subsystems of the given simulation problem or some logical and consistent computational block which is defined by a corresponding mathematical model. The software module for implementation of each block is defined as a task.

Definition 3. A *module/task deadline* represents the hard timing constraint imposed on the execution of each task. The corresponding time interval is equal to the discretization period and depends on the module dynamic, required accuracy, and applied numerical methods. The module/task deadline cannot be violated in the real-time simulation process by any of the tasks.

Definition 4. A *module/task workload* represents the amount of computation necessary for module equation processing, normalized to the module/task discretization period, i.e., the deadline interval. The formal definition of module/task workload is given by

$$W_{\mathrm{M}}^{(i)} = \left(\frac{1}{T_i}\right) \cdot C_i, \quad i = 1, \ldots, n, \tag{1}$$

where T_i and C_i represent the discretization and implementation period for ith module/task.

Definition 5. A *global workload* represents the amount of computation necessary for real time simulation of all modules/tasks which constitute mathematical model which is the objective of real-time simulation. The formal definition of global workload is given by

$$\sum_{i=1}^{n} \left(\frac{1}{T_i}\right) \cdot C_i. \tag{2}$$

A large number of dynamic modules which participate in typical training or testing scenarios very often cause that the global/system workload is characterized by

$$m < \sum_{i=1}^{n} W_{\mathrm{M}}^{(i)} < m + 1, \quad m > 1. \tag{3}$$

This means that the real time simulation of such a system requires at least $m + 1$ processors.

According to notation used in [9] the task set which is going to be scheduled is consisted of n tasks $\tau_1, \ldots, \tau_n$, each of which is represented by an ordered pair (C_i, T_i), where C_i is the run-time, i.e., implementation time and T_i is the request period or deadline. By using this information, appropriate real-time scheduling algorithm makes its decisions explicitly on the temporal characteristics of the task set. Using these parameters, the level of computational granularity and the assignment and scheduling policy can be adapted to different workload characteristics.

2.1. Clustering assignment and scheduling policy

Modules/tasks with workload specification characterized by

$$W_{\mathrm{M}}^{(i)} = \left(\frac{1}{T_i}\right) \cdot C_i < 1, \quad i = 1, \ldots, n_{\mathrm{C}}, \tag{4}$$

are candidates for clustering-assignment policy, since they require only a fraction of a single processor frame time. Taking into account the impressive computing power of modern signal processors, such as TMS320C30/C40 or AD21020/60, many dynamic modules or subsystems which participate in a real-time simulation scenarios have workload parameter significantly less than 1, and therefore, can be clustered together and assigned to the same processor. Time metrics for this policy are characterized by multiple deadlines because each subsystem is processed with sampling time, i.e., deadline in accordance with its dynamic features. Task communications for this policy are characterized by predominantly local, i.e., innerprocessor communication.

As every other scheduling policy, clustering scheduling policy must meet the deadlines assigned to each task in a cluster. In order to be schedulable, the task set or cluster must satisfy the following inequality:

$$\sum_{i=1}^{j\leqslant nc} \frac{C_i}{T_i} + \sum_{i=1}^{j\leqslant nc} \frac{T_{\mathrm{IO}}^{(i)}}{T_i} + \sum_{i=1}^{j\leqslant nc} \frac{T_{\mathrm{Sch}}^{(i)}}{T_i} \leqslant 1. \tag{5}$$

Since task communications are predominantly local the second term in (5) can be neglected. It was shown in [7] and it will be discussed later, that task switching time when clustering scheduling policy is implemented on TMS320C30/C40 is almost 1000 times smaller than deadlines of typical dynamic systems. Therefore, the previous expression can be rewritten as follows:

$$\sum_{i=1}^{j\leqslant nc} \frac{C_i}{T_i} \leqslant 1. \tag{6}$$

The special feature of the clustering scheduling policy is introducing *basic cycle* time T_{bc}, which is the *greatest common divisor* of all deadlines of the related task set. By multiplying both sides of (6) with T_{bc}, the following is obtained,

$$\sum_{i=1}^{j\leqslant nc} \frac{T_{\mathrm{bc}}}{T_i} \cdot C_i \leqslant T_{\mathrm{bc}}, \tag{7}$$

which is the schedulability condition for clustering scheduling policy. This policy assigns to each task the amount of processing time per basic cycle T_{bc} which is equal to

$$\frac{T_{\mathrm{bc}}}{T_i} \cdot C_i. \tag{8}$$

Unlike rate-monotonic scheduling policy [9,10], for this policy all tasks in the set are of equal priority. The task with greater request rate will in accordance with (8) get the greater part of basic cycle time. After T_{bc} elapses, the scheduling sequence is repeated in the same manner again. At the same time, the order of task execution over the basic cycle interval is of no importance. Using standard classification [3,9,10,14], clustering scheduling algorithm can be classified as static pre-emptive scheduling policy. Although all tasks have the same priority and order of execution is chosen arbitrarily, once the execution is started the predefined order of execution and pre-emption is not changed. This significantly simplifies software implementation of the clustering scheduling algorithm enabling small task switching time and is contrary to the deadline driven, i.e., dynamic scheduling algorithms [3,9,10], such as earliest deadline, least-laxity first and maximum-urgency-first algorithms. On the other hand it is possible to attain almost 100% of utilization factor, which is the prime feature of dynamic scheduling algorithms. Although in [10] it was stated explicitly that rate-monotonic priority assignment is optimum in the sense that no other fixed priority assignment rule can schedule a task set which cannot be scheduled by the rate- monotonic priority assignment, the following example will show that this statement is not true [1]. Consider the four tasks set:

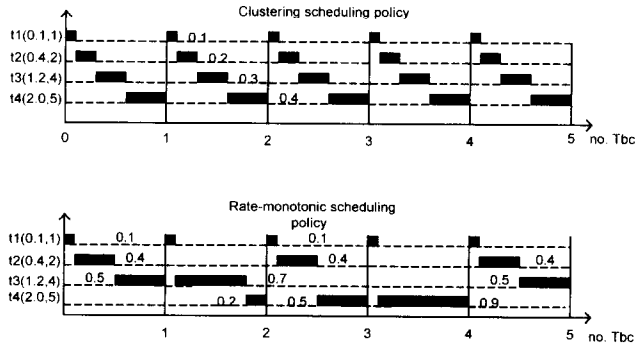

Fig. 1. Clustering and rate-monotonic scheduling policies.

$\tau_1 = (0.1, 1)$, $\tau_2 = (0.4, 2)$, $\tau_3 = (1.2, 4)$, $\tau_4 = (2.0, 5)$. The parameters in brackets determine the tasks run-times and deadlines, respectively. According to the given definition $T_{bc} = 1$, and the amount of processing time per basic cycle is computed according to expression (8). The application of clustering and rate-monotonic scheduling policies to the presented task set is illustrated in Fig. 1.

According to the schedulability test for rate-monotonic scheduling policy [3,9,10], given task set is not schedulable under this policy. Fig. 1 illustrates this fact. After 5 time units, τ_4 has yet not been completed. Although the order of execution for clustering scheduling policy is of no importance, it was intentionally given the same priority assignment like the rate-monotonic assignment policy. As can be seen from Fig. 1, the chosen priority scheme is fixed and the task set is still schedulable. This, however, is not true for rate-monotonic scheduling algorithm.

The clustering assignment and scheduling policy is applied on the satellite attitude control system [6,7]. The mathematical model of satellite attitude control system dynamic is composed of four subsystems according to Fig. 2. Those subsystems are treated as tasks in accordance with the previously described methodology. The discretization time or deadline of each subsystem/task corresponds to the time constants or dynamics of the related subsystem/task. The task run times and deadlines are given in milliseconds. The rate and position sensors were integrated using AB-2 algorithm, the satellite equations of motions using modified Euler integration algorithm [6], and actuator dy-

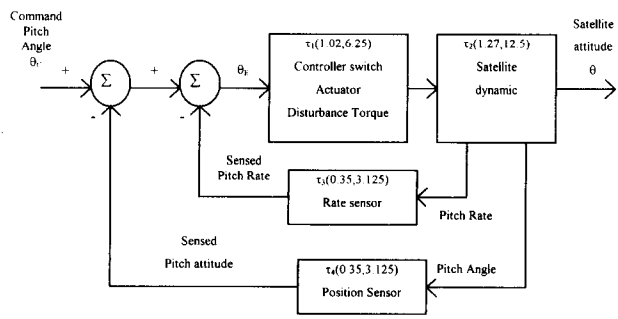

Fig. 2. Clustering scheduling policy applied to satellite attitude control system.

namic using the state transition method. The complete model was implemented on the TMS320C30 based DSP card. The clustering partitioning and scheduling policy was applied on the related four task set. All tasks were coded as *C* functions of type void. The object code of assembled mathematical model was linked with the scheduler object code. The information about execution time per basic cycle for each task, in accordance with clustering scheduling policy and expression (8), was reported to the scheduler in the form of the number of timer ticks. This was achieved by using integer array with a reserved name. The clustering scheduling policy was implemented as timer interrupt driven software kernel. Variables necessary for task communication were declared as global. The worst case delay, introduced by scheduler during context switching, was 130 CPU cycles of the TMS320C30 signal processor [7]. For 30 MHz version of C30 it means 8.66 μs. A similar or even better feature can be obtained on AD21020 or AD21060 signal processors. For faster CPU clocks the scheduler delay approaches 4–5 μs, which can be neglected in relation to the task deadline since sampling times of a majority of dynamic systems are in the millisecond range.

The use of the clustering scheduling policy enables that each module be processed in real time corresponding to the module dynamics, assuming schedulability of the related tasks set with maximum utilization factor. Additionally, the use of explicit integration methods (AB-2, modified Euler, state-transition method) increase the level of computational parallelism. Since I/O variables are declared global and are always available, because of the use of explicit integration methods, the computation is being performed asynchronously, i.e., parallel. Although the numerical complexity of the given example is low, it clearly illustrates the advantages of the simultaneous application of the clustering scheduling policy and explicit integration methods in real time simulation of the satellite attitude control system, and of control systems in general.

2.2. *Partitioning assignment and scheduling policy*

Modules/tasks with workload specification characterized by

$$W_{\mathrm{M}}^{(i)} = \left(\frac{1}{T_i}\right) \cdot C_i > 1, \quad i = 1, \ldots, n_{\mathrm{P}}, \tag{9}$$

are candidates for a partitioning-assignment policy, since they require multiple processor units to satisfy the real time constraints.

The basic property of the partitioning-assignment policy is characterized by the assignment of multiple processors to one module, since such policy will be applied to modules with a high workload. With respect to time metrics, this policy is characterized by a single deadline, which must be met by all subtasks generated by the partitioning procedure. With respect to communications, this policy is characterized by a predominantly interprocessor communication.

The partitioning assignment policy splits a module with high computational intensity into a set of subtasks, which distributed on the multiprocessor resources enable real-time simulation. The number of processors actually assigned to such a module depends on: the workload intensity of the balancing of the assigned computations, the degree of coupling between equations distributed among different processors and the efficiency of

interprocessor communications. If differential equations assigned to the same processor are tightly coupled to each other, at the same time loosely coupled with differential equations assigned to other processors and if such allocation is well balanced, the speed up factor can approach the ideal maximum.

The natural candidates for application of the partitioning assignment and scheduling policy are modules or systems described by partial differential equations (PDEs). The following example illustrates application of the partitioning assignment and scheduling policy and parallel processing platform, based on modern DSPs such as TMS320C40 or ADSP21060, to the solution of the 3-D stationary transport model [8] described by

$$-D\Delta C(X) + V(X)\nabla C(X) + \kappa C(X) = q(X). \tag{10}$$

The above model is supplied with appropriate boundary conditions at the free surface, coastal boundary and bottom. The equation describes the steady state process induced by an input substance distribution in some bounded volume or basin. Eq. (10) was applied [8] to the analysis of the lead distribution in the Punat Bay of the Adriatic Sea, the area of which is 2.4 km^2. Here $C(X)$ is the concentration field, $V(X)$ is the current field, κ is the extinction rate and $q(X)$ is the input distribution. ∇ is the gradient operator, denoting the derivatives with respect to coordinates x, y and z, and Δ is the Laplacian $\Delta = \partial^2/\partial x^2 + \partial^2/\partial y^2 + \partial^2/\partial z^2$. To simplify computation, the basin is divided into n layers. The concentration in each layer is $C_k(x, y)$, $k = 1, 2, \ldots, n$. The general form of the discretization scheme of Eq. (10) is

$$\begin{aligned} &L \bullet C_1 + I \bullet C_2 = q, \\ &L \bullet C_k + I \bullet C_{k+1} = U \bullet C_{k-1}, \qquad k = 2, 3, \ldots, n-1, \\ &L \bullet C_n + pC_n = U \bullet C_{n-1}, \end{aligned} \tag{11}$$

where L, U and I are matrices and p is a scalar. In this way the 3-D problem (10) is replaced by n 2-D coupled problems. It has been shown that satisfactory results could be achieved with $n = 4$ layers. To achieve required accuracy, the computation must be done in an iterative manner. It turned out that a dozen iterations is adequate to get an acceptably accurate solution. In the case of $n = 4$, the discretization scheme is given by (12), where the superscript ($i = 1, 2, \ldots, 11, 12$), denotes the number of the iteration and the subscript denotes the number of the layer.

$$\begin{aligned} &L \bullet C_1^i + I \bullet C_2^{i-1} = q, \\ &L \bullet C_2^i + I \bullet C_3^{i-1} = U \bullet C_1^i, \\ &L \bullet C_3^i + I \bullet C_4^{i-1} = U \bullet C_2^i, \\ &L \bullet C_4^i + pC_4^i = U \bullet C_3^i. \end{aligned} \tag{12}$$

By using four processors, such as the TMS320C40, each processor can perform one iteration. The implementation of Eq. (12) together with appropriate CPU interconnections is shown on Fig. 3.

The problem is characterized by the following features. Each 2-D layer has 5000 nodes. For solving all four layers the amount of $2.25 \cdot 10^8$ multiplication/additions and

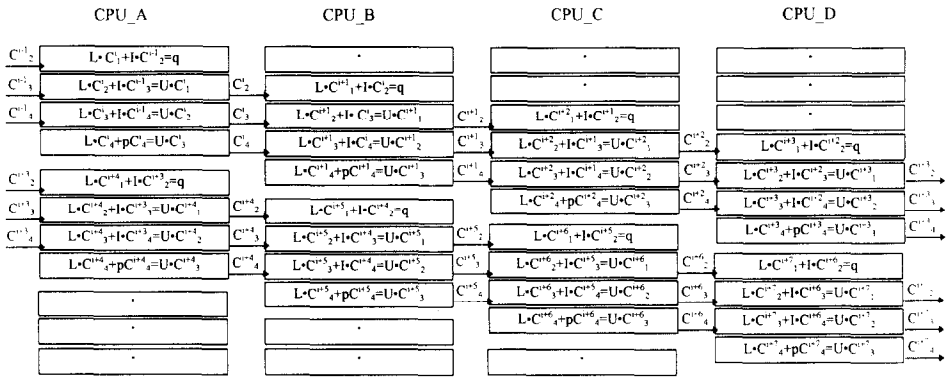

Fig. 3. Four-processor implementation of the PDE discretization scheme.

$1.5 \cdot 10^6$ divisions are required. One TMS320C40 can perform the required number of operation in about 11 seconds. While single processor implementation requires 132 seconds, implementing each iteration on one processor according to Fig. 3, it would be possible to perform all necessary computations in 41–42 seconds. The various number of different scenarios corresponding to the various current fields can be evaluated in a few minutes, which is very important for the decision making process. The average communication requirements of $8 \cdot 10^5$ bit/s are easily satisfied since each TMS320C40 DMA channel assures a bi-directional communication rate of up to 20 Mbytes/s. The main difficulties are enormously big memory requirements, 40–60 MWords per one processor.

Time metrics in the partitioning assignment policy are characterized by a single deadline, and therefore, the partitioning scheduling policy is completely determined by the assignment policy. Modules/tasks with workload specification characterized by

$$W_{\mathrm{M}}^{(i)} = \left(\frac{1}{T_i}\right) \cong 1 \quad i = 1, \ldots, n_{\mathrm{S}} \ (n_{\mathrm{C}} + n_{\mathrm{P}} + n_{\mathrm{S}} = n) \tag{13}$$

require a single processor unit for real-time simulation. Therefore, such modules are assigned to one processor.

On the basis of this assignment and scheduling policy, a global workload which constitutes modules with a different workload intensity, has to be distributed on a set of signal processors according to the following formula,

$$W_{\mathrm{DSP}}^{(k)} = W_{\mathrm{IO}}^{(k)} + W_{\mathrm{Sch}}^{(k)} + \sum_{i \in S_k} W_{\mathrm{M}}^{(i)} \cong 1, \quad k = 1, 2, \ldots, l, \tag{14}$$

where l is the number of processors which will participate in a problem solution and generally is $l \geqslant m+1$, S_k is some partition of set $\{1, \ldots, n\}$ that includes all modules/tasks which constitute a simulation problem, $W_{\mathrm{IO}}^{(k)}$ determines workload of interprocessor communications of processor k, and $W_{\mathrm{Sch}}^{(k)}$ determines workload for task scheduling policy of processor k.

The efficiency of pre-run-time assignment and scheduling policy can be measured by a multiprocessor resource utilization factors such as load imbalances of processors and

by a ratio between scheduling time length and task computation times. On the basis of these criteria, some algorithms for automatic load balancing can be established. The main advantages of the pre-run-time scheduling strategy in hard real time simulations are: meeting all real time constraints and minimization of I/O communications and context switching time.

Minimization of the number of signal processors needed for real-time simulation of a given problem requires minimization of processors times for interprocessor communications and scheduling policy, i.e.,

$$T_{\mathrm{IO}}^{(k)} + T_{\mathrm{Sch}}^{(k)} \rightarrow \min. \tag{15}$$

Using a very fast clustering scheduling policy, implemented on the signal processor TMS320C30/C40 [7], and asynchronous interprocessor communications based on parallel I/O ports and DMA coprocessors of TMS320C40, expression (15) can be minimized. Consequently, the speed up factor in multiprocessor real time simulation using C40 signal processors can achieve nearly ideal linear approximation.

3. Parallel processing development system with four TMS320C40s

The main real-time system requirements for efficient multiprocessor real time simulations are: multiple FIFO buffered I/O bi-directional channels, concurrent I/O and CPU processing, direct high speed interprocessor communications controlled by DMA coprocessors, fast context switching, multiple busses, multiple memories, and powerful arithmetic units. The TMS320C40 is one of the modern DSPs which satisfies the listed requirements [16].

The primary benefit of the DMA coprocessor is to maximize sustained CPU performance by completely alleviating the CPU of burdensome I/O duties. DMA transfers data to and from any C40 memory mapped location (on-chip memory, external memory, input and output communication port registers, etc.), even routes data directly from one processor to another processor across communication ports, without interfering with the operation of the CPU.

In this research, practical experimentation has been done using Texas Instruments parallel-processing development and debug tools. The research goal was to show that communication overhead specific for partitioning assignment policy can be almost neglected when real time simulation problem is implemented on the multiprocessor platform composed of modern signal processors such as TMS320C40. Although the demonstration example can be considered trivial in order to illustrate the previous statement, the eight level FIFO buffer on each DMA communication channel supports implementations, which are considerably demanding from the interprocessor communication point of view. The development tools provide easy-to-use real-time emulation and software development for parallel-processing architectures. These tools include: XDS510 parallel in-system emulator capable of debugging any number of processors in a system and *PPDS*. A block diagram of this system is shown in Fig. 4.

The key features of the PPDS are: four on-board C40 parallel processors supported by a local bus consisting of 64 K × 32-bit words of zero wait-state SRAM and 8 Kbytes

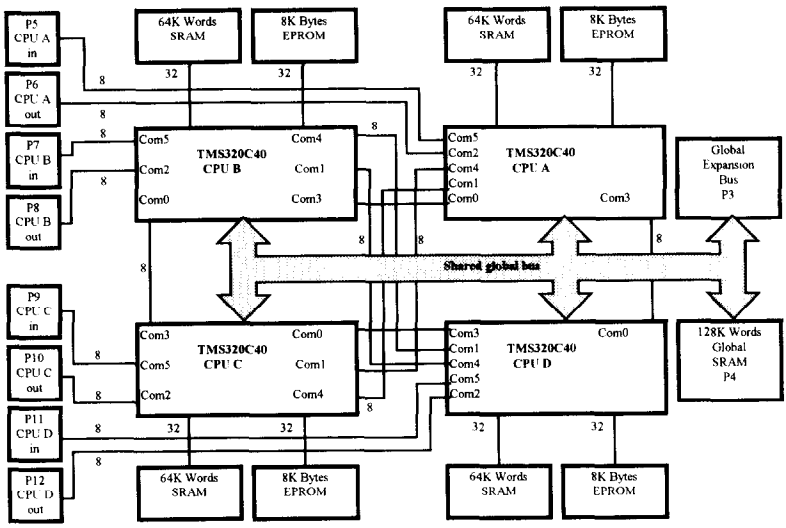

Fig. 4. TMS320C40 PPDS block diagram.

of EPROM; 128 K × 32-bit words of one wait-state SRAM on a shared global bus; an expansion bus connector that provides an external interface to the shared global memory bus; direct connections between each C40s through the communication ports; eight external communication connectors that provide an interface for connecting off-board C40s and external peripherals to the PPDSs C40s; a JTAG test connector that provides an interface for connecting the XDS510 into a single ring with C40s allowing debugging of all processors through a single interface. The XDS510 provides a separate debugger for each C40, allowing debugging of individual C40s in C, assembly, or both simultaneously. The XDS510, and thus the PPDS, is used with IBM-compatible PCs using OS/2. The OS/2 must be used for the multitasking environment in order to simultaneously debug multiple processors.

4. Asynchronous parallel interprocessor communications in real time simulation

In order to illustrate the asynchronous interprocessor communications (inherent for partitioning-assignment policy) using parallel I/O ports and DMA coprocessors, let us consider the following four-task–four-processor real time simulation problem.

$Task_1$ is defined by Eqs. (16) and represents an input signal with discretization period of 15 μs.

$$u(t) = \sin(2000\pi t) + \sin(7000\pi t). \tag{16}$$

With discretization period of 15 μs the 133 points $(t_i, u(t_i))$ of 2 ms-periodical function will be generated. $Task_{2,3,4}$ are defined by Eqs. (17)–(19)

$$\frac{y_1(z)}{u(z)} = \frac{0.53249z^2 - 0.92657473z + 0.44088439}{z^2 - 1.914135668z + 0.960945215}, \tag{17}$$

$$\frac{y_2(z)}{y_1(z)} = \frac{0.3547z^2 - 0.59510103z + 0.282879754}{z^2 - 1.82317966z + 0.865658384}, \tag{18}$$

$$\frac{y_3(z)}{y_2(z)} = \frac{0.22671z^2 - 0.378275279z + 0.19988579482}{z^2 - 1.64111158z + 0.688403953}, \tag{19}$$

which represent lowpass digital Chebyshev filter sections obtained from an analog prototype using MSI method [2] with $T = 15\ \mu s$. The discretization period of 15 μs is equal to 240 clocks (one clock is 62.5 ns). Using the C Source Debugger [19] the implementation times of $Task_{1,2,3,4}$ have been precomputed using RUNB command, which measures the number of clocks between two breakpoints. The measurement results obtained by C Source Debugger show that 181 clocks for $Task_1$ and 183 clocks for other tasks are required. On the basis of this information, it is easy to precompute module/task workloads for all tasks as well as the global workload. For the discretization period of 15 μs the workloads of the related tasks were $W_M^{(1)} = 0.754$, $W_M^{(2)} = W_M^{(3)} = W_M^{(4)} = 0.7625$. Therefore, it can be written as follows:

$$\begin{aligned} &W_M^{(i)} + W_M^{(j)} > 1 \quad i,j = 1,2,3,4,\ i \neq j \\ &3 < W_M^{(1)} + W_M^{(2)} + W_M^{(3)} + W_M^{(4)} < 4. \end{aligned} \tag{20}$$

On the basis of these equations, it is evident that the real time simulation of the given problem requires at least four processors. If we take into account TMS320C40 abilities for asynchronous I/O communications, i.e., simultaneous CPU and I/O operations, that leads to $W_{IO}^{(k)} \to 0$, and the fact that there is no need for tasks clustering and scheduling overhead in this case, we can write $W_{IO}^{(k)} + W_{Sch}^{(k)} \to 0$ for $k = 1,2,3,4$. The relatively small discretization period of 15 μs is chosen intentionally to show that interprocessor communication time when using concurrent DMA channels can be neglected even at such high processing rates. This is, in a way, equivalent to the case with significantly lower processing rates, but with more demanding interprocessor communication requirements as it is the 3-D stationary transport problem described by Eqs. (10)–(12) and Fig. 3. Therefore, optimal task assignment policy in this example is given by one-task–one-processor and is shown in Fig. 5.

The four-tasks–four-processor example has been implemented on PPDS TMS320C40 platform using C-language supported by runtime support library [17] and parallel runtime support library [18]. Communication between PC and the PPDS TMS320C40 platform is supported by the OS/2 operating system and C Source Debugger [19]. DMA channels have been prepared for split mode and autoinitialization transfer. The

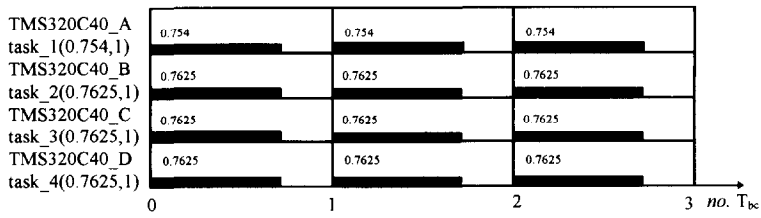

Fig. 5. Task distribution across CPU's for parallel task execution.

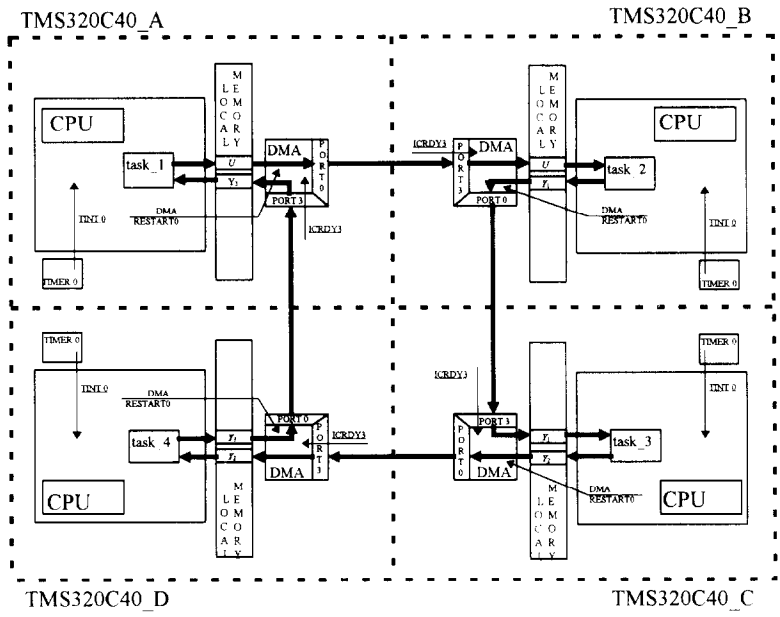

Fig. 6. Real time simulation using PPDS TMS320C40 platform.

split mode transforms one DMA channel into two DMA channels; *primary* and *auxiliary channel*. The primary channel is dedicated for reading data from a location in the memory and writing it to an output communication port. The auxiliary channel is used for receiving data from an input communication port and writing it to a location in the memory. The DMA coprocessor controls and executes all processes concerning data transfer and autoinitialization. In order to start I/O communications, CPU has only to write bits 11 in DMA channel control register. As a consequence, output data transfer begins using the primary channel. The parallel start of all four processors is achieved with assembly routine *syncuont.asm* [12]. After simultaneous start the processors begin with the real-time simulation under the control of timer interrupts. In interrupt routine each processor reads DMA local memory input value, computes new output values by using recursive equations of (16)–(19), starts DMA coprocessor and waits a new interrupt. Auxiliary and primary DMA channels perform data transfer between designated PORTS and memory locations, according to Fig. 6. This figure illustrates implementation of the four task example on the PPDS platform.

The timing diagram of this simulation scenario is shown by Fig. 7. From it is evident that: $Task_{1,2,3,4}$ are activated by interrupt signal TINT and executed simultaneously, DMA auxiliary channels are activated by the "input channel ready" signal ICRDY, input and the output data are transferring under the control of the DMA coprocessor and CPU performs computations on the input data provided by DMA coprocessors in local memory without checking if they have been received or not.

From recursive equations of (17)–(19) it is evident that the values $u(n+1)$, $y_1(n+1)$, $y_2(n+1)$ are not available at the time interval $(n : n+1)$ when computing output variables $y_1(n+1)$, $y_2(n+1)$, $y_3(n+1)$. To ensure the compatibility of real time

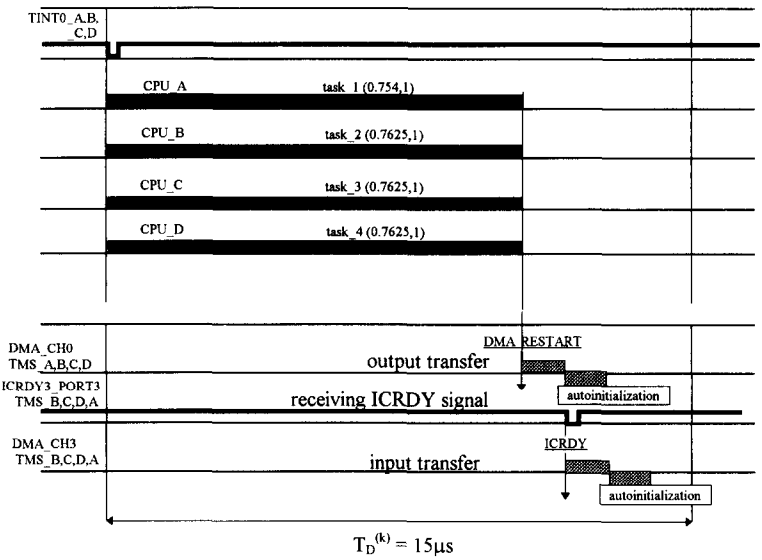

Fig. 7. Timing diagram of asynchronous CPU and DMA operations.

inputs without introducing time delay we were forced to use second-order extrapolation formula based on nth, $(n-1)$st and $(n-2)$nd values of u, y_1, y_2 [4,5]. The extrapolated values were introduced instead of $u(n+1)$, $y_1(n+1)$, $y_2(n+1)$ in recursive equations of (17)–(19). Due to small discretization period of 15 μs the sum of squares of deviation normalized on one discretization point was $S = 4.18 \cdot 10^{-5}$, which can be considered as satisfied accuracy.

5. Conclusions

The predictable task execution and deterministic response time are necessities in all hard real-time applications with stringent timing constraints. Therefore, a predictable deterministic computing algorithm based on the module/task constraints and new static allocation and scheduling algorithm have been introduced. This new pre-emptive static algorithm called clustering scheduling policy is intended for scheduling hard deadline periodic tasks. It approaches 100% of utilization factor like dynamic deadline driven scheduling algorithms, while at the same time remains easier for software implementation like other fixed priority or static scheduling algorithms. Furthermore, the speedup factor in real time simulation using multiprocessor resources strongly depends on the architecture of the multiprocessor system, especially on the interconnection between parallel processors. Therefore, special attention has been paid to the asynchronous I/O communications. Parallel signal processors TMS320C40 allow simultaneous CPU and I/O operations using high-speed 8 bits communication ports and 6-channels DMA coprocessor. DMA coprocessor maximizes sustained CPUs performance providing continuous data transfer between processors over I/O ports without any CPU intervention. Architecture of this processor is ideal for use in multiprocessor real-time simulation applications,

such as hardware-in-the-loop testing, man-in-the-loop training and implementation of complex tactical trainers, which has been clearly illustrated.

Acknowledgement

The authors would like to thank Dr Nedžad Limić from the Department of Mathematics, University of Zagreb, for his suggestions related to the application of the partitioning assignment and scheduling policy on the example of the 3-D stationary transport problem.

References

[1] K.K. Ćosić and R.M. Howe, Real time simulation using multiprocessor resources, in: *Proceedings of the 1991 European Simulation Multiconference*, Copenhagen, Denmark.
[2] K. Ćosić and I. Kopriva, Some improvements of the state transition method in real-time simulation of linear systems, *Trans. Soc. Comput. Simulation* **11** (1) (1994).
[3] A.D. Ferrani, Real-time scheduling algorithms, *Dr. Dobb's J.* (December 1994).
[4] A. Haraldsdottir and R.M. Howe, *Multiple Frame Rate Integration* (American Institute of Aeronautics and Astronautics Inc., 1988).
[5] R.M. Howe, Dynamics of real-time digital simulation, *Course Notes*, Ann Arbor, MI, 1988.
[6] R.M. Howe and W.P. Kavanough, AD 100 simulation of a satellite attitude control system, ADI Application Rept., Ann Arbor, MI, 1989.
[7] I.Kopriva, K. Ćosić, A new approach to the low cost hardware-in-the-loop simulation, in: *Proceedings of the 37th International Conference Corema*, Croatia (1992) 431–435.
[8] T. Legović, N. Limić and V. Valković, Estimation of diffuse inputs to a coastal sea: Solution to an inverse modeling problem, *Estuarine, Coastal Shelf Sci.* **30** (1990) 619–634.
[9] C.L. Liu and S.K. Dhall, On a real-time scheduling problem, *Oper. Res.* **26** (1) (1978).
[10] C.L. Liu and J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, *J. ACM* **20** (1) (1975) 46–61.
[11] O.A. Palusinski, Simulation of dynamic systems using multirate integration techniques, *Trans. Soc. Comput. Simulation* **2** (4) (1985).
[12] R.M. Piedra, A parallel approach for solving matrix multiplication on the TMS320C4x, DSP Application Rept. (lit. number SPRA026), Texas Instruments Inc., 1991.
[13] L. Sha and J.B. Goodenough, Real-time scheduling theory and Ada, *IEEE Comput.* (April 1990) 53–62.
[14] B. Sprunt, L. Sha and J. Lehoczky, Aperiodic task scheduling for hard-real-time systems, *J. Real-Time Systems* **1** (1) (1989) 27–60.
[15] J.A.Stanković, Misconceptions about real-time-computing, *IEEE Comput.* (October 1988) 10–19.
[16] TMS320C4x User's Guide (lit. number SPRU036), Texas Instruments Inc., 1991.
[17] TMS320 Floating-Point DSP Optimizing C Compiler User's Guide (lit. number SPRU034), Texas Instruments Inc., 1991.
[18] TMS320C4x Parallel Runtime Support Library User's Guide (lit. number SPRU084), Texas Instruments Inc., 1992
[19] TMS320C4x C Source Debugger User's Guide (lit. number SPRU054), Texas Instruments Inc., 1992.