DESIGN AND TESTING OF HOMOGENOUS SINGLE BUS TIGHTLY COUPLED MULTIPROCESSOR SYSTEM FOR REAL TIME SIMULATION

INFORMATICA 3/89

Keywords: multiprocessor system, real time simulation, single bus, parallelization of mathematical models

Krešimir Ćosić, Ivan Miler i Igor Rašeta
VVTŠ KoV JNA Zagreb

ABSTRACT - The real time simulation for hardware or man in the loop testing presents the cost effective way for design, development, modification and testing of a complex and sophisticated weapons and industrial systems. This simulation technology is a constant challenges for most powerful computer systems. Therefore one short chronological review of computer architecture for time critical real time simulation is given. For such kind of application one homogenous single bus tightly coupled multiprocessor system based on 8086/8087 single board computers has been designed. Furthermore, this article presents one concept for parallelization of mathematical models given by ordinary differential equations in real time environment. Simulator design for one spinning missile system, according to the accepted procedure, illustrate the abilities of realized multiprocessor simulator system.

SAŽETAK - Simulacije u realnom vremenu za testiranje realnog hardvera ili operatora u zatvorenoj petlji, predstavljaju efikasan put za projektiranje, razvoj, modifikaciju i testiranje kompleksnih sofisticiranih vojnih i industrijskih sistema. Ovakva simulaciona tehnologija predstavlja stalan izazov za najsnažnije računarske sisteme. Iz tog razloga dan je jedan kratak kronološki pregled računarskih arhitektura za vremenski kritične simulacije u realnom vremenu. Za takvu vrstu primjena, u okviru ovog rada, realiziran je jedan homogeni, čvrsto spregnuti multiprocesorski sistem s jednom sabirnicom i nizom procesorskih ploča baziranih na procesorima 8086/8087. Pored toga, izveden je i prikazan jedan koncept za paralelizaciju matematičkih modela zadanih običnim diferencijalnim jednadžbama. Projektiranjem simulatora, prema usvojenoj proceduri za jednu rotirajuću raketu, prikazane su mogućnosti realiziranog multiprocesorskog simulatora.

1. INTRODUCTION

The increasing complexity and sophistication of modern process control and weapon systems has established a category of real-time simulation which uses hardware components integrated in the process of simulation. This kind of simulation, so-called hardware-in-the-loop (HIL) simulation technology, has proved to be a very cost effective method in design, development, modification, and testing of complex weapons and industrial systems [1,2,3,4]. In HIL simulations, for example, adequate computer equipment can be used to simulate the aerodynamics and flight equations of the missile, while the real hardware subsystem such as RF sensors, IR sensors, fin actuators, autopilots, guidance and homing on board computers can be embedded and used for design and testing of a closed-loop system. In this case, HIL simulation provides a reproduction of what the real hardware subsystem (missile seeker) really processes in real environment, on the basis of simulation of the missile aerodynamics, flight equations and targets movement simulated by suitable pseudo-target generator. In a this way it is possible to perform nondestructive testing, verification and validation of actual missile or process control subsystem in near realistic environments. This preflight check and similar industrial testing in the early phase of design and development provides effective way to analyze the overall performance capability of the closed loop system and to predict a performance at minimal cost.

2. COMPUTER ARCHITECTURE FOR TIME CRITICAL REAL TIME SIMULATIONS

Hardware-in-the-loop and man-in-the-loop simulations, which require time critical real-time simulations have proved to be constant challenges for the most powerful computer systems. Demands for more and more fidelity and accurate simulation of dynamic system characterized by ordinary differential

equations, significantly increase demands for additional speed and power from simulation computers. These demands have increased at the rate at least as fast as the rate of development and improvement in computer technology. Thus, today, available simulation capabilities have the same relationship to the requirements as 10 years ago [3]. Furthermore the speed requirements of the more challenging simulation applications, very frequently exceed the capabilities of even the most powerful mainframe computers. During the 1960's hardware-in-the-loop simulation of time critical processes depended on analog computers, such as EAI 231, EAI 781. In analog computers parallel operations of many computing elements provide very high speed of processing which is the most significant for time critical real-time simulation. Programming of these processors was a manual process using the patch boards and fixed point scaled equations. In early 1970's hardware-in-the-loop simulations has predominantly shifted to hybrid computations i.e. combination of analog and digital hardware, such as EAI PACER 100, to provide the required computational capabilities. In the middle of 1970's with the advent of fast mainframe digital computers the emphasis in application is based exclusively on digital hardware. But at that time the speed requirements of more challenging simulations frequently exceeded the capabilities of even the most powerful mainframe computer such as IBM 360, CDC 7600, Univac 1108 and so on. Today, currently available mainframe supercomputers such as CRAY-1, CRAY X-MP-1, CRAY X-MP-2, IBM 3090/VF-200, NEC SX-1E, NEC SX-2, CDC CYBER 205, Amdahl 1200, Hitachi S-810/20 and so on, in majority of cases provide necessary computational power, but very often doesn't provide the cost effective approaches for such applications. Therefore in the late 1970's the trend in architecture of a digital computer system for time critical real-time simulations was toward the more specialized architectures. The first of these devices was peripheral array processor AD-10 (1979) manufactured by Applied Dynamics Inc. It was simulation-oriented peripheral processor intended primarily for the simulation of systems of ordinary differential equations. Performances of this system are given in Table 1. They are related to estimation of computer power necessary for development of helicopter simulator.

Impressive speed of the machine when applied to ordinary differential equations results from its advanced technology which provides very high processing speeds and by the extensive pipelining and parallelling and from specialized computing and memory units suitable for solving nonlinear ODE-s [7]. New version of this system AD-100, which is characterized by significant improvement in performance (Table 2) has appeared on the market in 1984.

TABLE 1

HELICOPTER SIMULATORS [5]

computer	and achieved frame times		
CYBER 175	45 ms	FPS AP-120B	4.5 ms
CDC 7600	15 ms	AD-10	0.8 ms

TABLE 2

MODEL OF A WHIRLING FLEXIBLE BEAM [6]

computer	and AD-100	advantage	
ADI AD-100	1.00	IBM 3033	7.45
CRAY 1-S	3.35	FPS 164	17.95
IBM 3081	5.40	HEP H-1000	36.65

But very often hardware-in-the-loop real-time simulation requires simulator systems that are more cost effective and portable. The first cost effective way for attaining analog computer speed leads to parallel operation of multiple digital microprocessors. However, the price/performance ratio of multimicroprocessor system was very attractive and therefore a number of attempts to interconnect relatively inexpensive general-purpose microcomputers have been made over the past years for designing a complex simulator systems. With the increasing availability of very fast and very economical single board computers, it becomes feasible to design the construction of network of microprocessors which will form special-purpose simulator. This approach was very attractive and more favorable in speed/cost ratios in relation to other solution. Therefore a number of microprocessor networks were developed for real-time simulation throughout 1980. The real-time multiprocessor simulator (RTMPS) project at the NASA Lewis Research Center for the simulation of jet engines (1984) was one of the first and most significant [8]. The recent introduction of powerful multiprocessor systems by a large number of vendors (1985-1987), such as Ametek Computer Research Division, Alliant, BBN Advanced Computers, Elxsi, Encore Computer, Flexible Computer, Intel Scientific Computers, Ncube, Thinking Machines and so on, has increased the interest of engineers and scientists in this approach to high speed real-time scientific computations. But it is necessary to maintain that this approach is not cost effective and quite attractive for the majority of customers and simulator vendors. Today, very cost effective approach to parallel digital real-time simulation is based also on the network of transputers [9,10]. The T800 Transputer contains a 10 MIPS 32-bit processor, on chip RAM, timer and I/O interfaces which are based on serial communication channels. T800 have four links per chip and they use a clock rate of 20 MHz on the serial link, so that the communication channel between two Transputers requires only the connection of two wires for the link. Two T800 Transputers with floating point hardware with a speed of 1-2 MFLOPS, in the simulation of 2-nd order system by the RK 4 integrators is

approximately twelve times faster than the Intel 80386/80287 and six times faster than the Motorola 68020/68881 [10]. Further evolution of transputer networks by Inmos Inc. and Micro Way provides abilities to design and build arbitrarily large parallel processing machines. The 32-transputer array has been used in simulator design for modelling the flow through a jet engine's turbine-blade cascade by the Rolls-Royce at Derby [11]. This model has required ten minutes to run on a 32-transputer array and two minutes on a Cray XMP-48. Since the Cray cost over 125 times as much as 32-transputer array, the price performance ratio is 25:1 in favor of the transputer network.

But parallel operation and parallelism doesn't guarantee performance, and may, in fact, limit it. Example for this is successful replacement of 64-processor system Illiac IV with higher performance serial processor Cray I. The number of processors, interprocessor communications, memory organization and numerous other factors interact to limit or to enhance processor performance. The effective utilization of a network of processing elements or microcomputers poses difficult scheduling and allocation problems. This means that the major difficulty in using parallel processor is the effective software support, so that the total performance improvement in relation to the serial processing is dependent in the same time on the numerical procedures which are used i.e. techniques of discretization, techniques for decomposition and then, on the power of hardware for simulation.

3. ARCHITECTURE OF THE REAL-TIME MPS-AMS MULTIPROCESSOR SIMULATOR

The real-time Multiprocessor Simulator - MPS organized on a shared single bus -AMS, shared dual-port memories i.e. on tightly coupled multiprocessor topologies [12] is shown in Figure 1. Modular design of this system provides a number of benefits which are related with its flexibility in modification, reconfigurration and maintenance [13]. Four Siemens 8086/87 based single board computers (SBC) AMS-M8-A8 are used to realize simulator hardware on principles of master/slave relationship. SBC's boards are connected through the 16-bits data bus AMS-M (European realization of the IEEE 796 Multibus I) which supports the real-time processing features. Access to the analog I/O world has been provided by the 16/32 channels analog to digital input board 12-bits AMS-230-A1, and by the four digital to analog channels 12 bits, realized on AMS-M596 standard bus interface board. Such system is characterized by low functional complexity, physical compactness, and relatively low-cost. Communication among the processors is performed via message passing in "mailboxes" that reside in distributed dual port memory. Access to this memory occurs via a single time shared bus.

In the phase of the configuration of real-time multiprocessor simulator, development of the simulation real-time software requires selection of the modules, their editing, compiling and linking on the host PC XT system. The following step in the procedure is related to the assembling modules according to the block diagram of simulated process or system and their testing, evaluation and validation. After that the tested modules are downloaded onto the master processor board and finally, created modules are mapped from the master boards to the slaves boards of MPS according to the accepted decomposition schemes [14].

Furthermore the host system provides the overall control of the simulation activities through the suitable graphic language. With additional multiuser microprocessor development system Tektronix 8560 i.e. through different integration station Tektronix 8540, this system enables efficient development of custom design hardware and software modules [15]. Two processors, host and master, communicate using interrupt system via a PC bus window, through the high speed SMP - PC interbus SMP-E570-A1. This technique is selected as the fastest available communication between the two processors, which allows one system to access the address on a companion system's bus as through the address on its own bus. To prevent conflicts in sending and receiving data between PC-XT and AMS system through PC memory, the synchronization mechanism uses the flag test-and-set procedure (semaphore) [16].

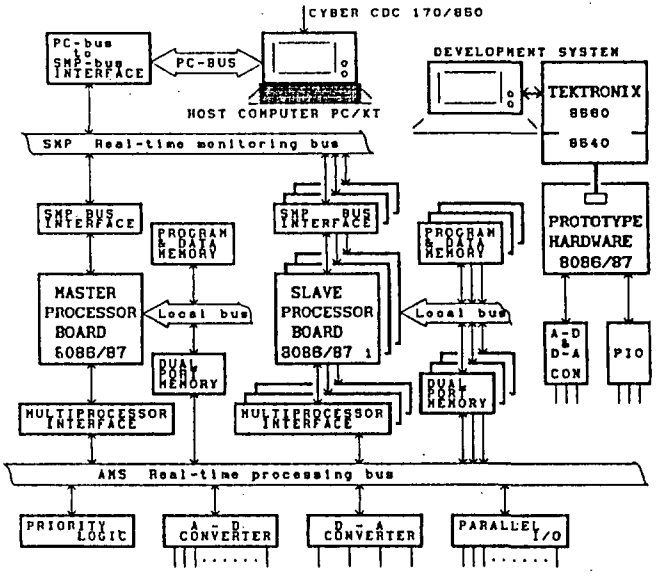

Figure 1. Multiprocessor architecture for Hardware-in-the-loop testing

AMS-M bus protocol defines master-slave communication and multiprocessor arbitration which is strongly problem dependent. A real-time monitoring SMP-M bus is 8/16 bits data bus architecture with its own bus interface on SBC board but without multiprocessor arbitration. The local bus on the AMS

boards connects the processors to all on board input/output devices (24 lines PIO 8255, RS 232 SIO 8251), local memory (EPROM 2764, SRAM 6116) and communication memory (dual port memory SRAM 6116), as shown in Figure 2. This bus permits independent execution of onboard activities.

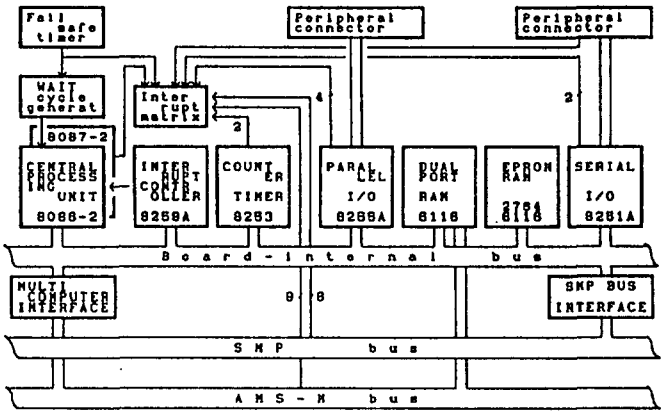

Figure 2. Single board computer AMS-M6-A8

A backplane provides the physical connections of AMS-M and SMP-M bus signals and priority resolver lines to set the priorities. Each board has a fixed priority which can be changed through jumpers on the backplane.

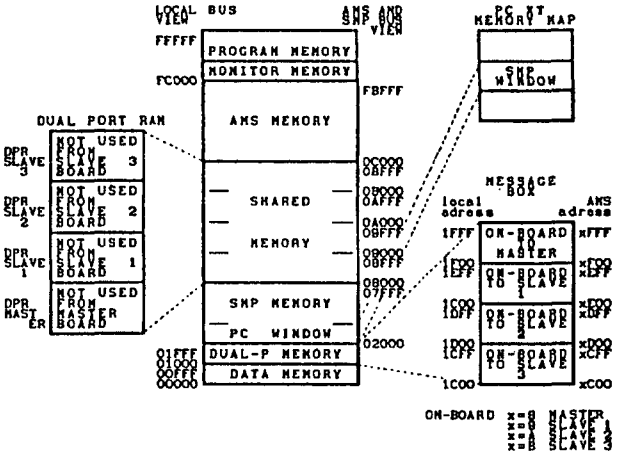

Figure 3. Multiprocessor memory organization

In this tightly coupled multiprocessor configuration, the processors communicate over the parallel bus, AMS-M, through a common i.e. shared memory. Generally speaking the common memory can be concentrated or distributed. In the accepted configuration, common memory is partitioned on each processor board as dual-port memory (DPM) to reduce bus occupation. Additionally, each processor board has a local memory which is especially interesting in loosely coupled decomposition algorithm which frequently use only local memory and seldom common memory. Furthermore, at the same time, this type of memory organization allows parallel access to shared memory without using the AMS-M real-time bus through the local bus. In addressing DPM all read accesses are local and do not use common bus. All write accesses use two different addresses. Onboard addresses are used to address its local memory and local dual-port RAM. Addressing DPM on the other boards are provided through AMS bus controller in memory space 0x000-0xFFF depending on the selected boards. The memory organization of this system is shown in Figure 3. To access the shared memory, it is necessary to gain the AMS-M bus, which then is locked-on through standard protocol.

The most important aspect of the bus interconnection topology used in this MPS is the bus arbitration technique. The priority level is determined by user through wrapping technic on backplane according to the Figure 4.

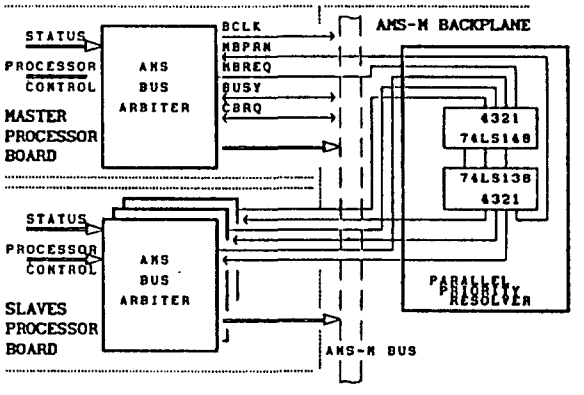

Figure 4. AMS-M bus arbitration

Each SBC has two arbitration lines, bus request (BREQ) and bus priority in (BRPN), which are used to gain access to the AMS bus. BREQ line comes from a SBC to priority resolver and indicates a request for control of the AMS bus. BPRN signal comes from priority resolver to a SBC and indicates that the processor may go ahead and use the bus since there is no other higher priority request for the AMS bus.

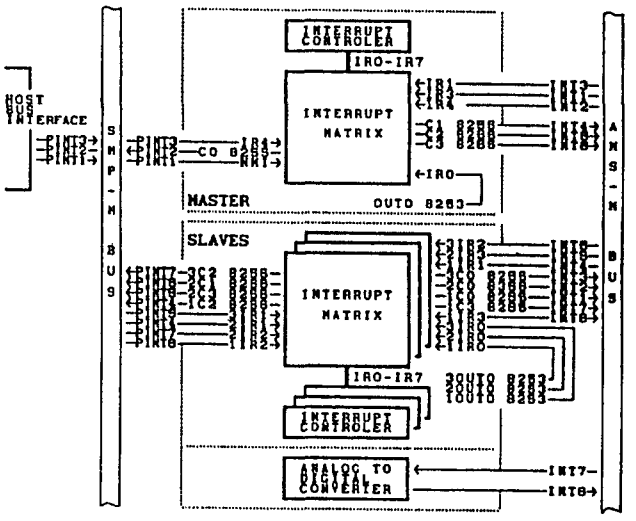

Figure 5. Interrupt system of MPS-AMS

To optimize the communications efficiency and to provide real-time processing of the real-time clock request, MPS system is predominately interrupt driven. The interprocessor communication begins by passing an interrupt request signal from one processor to another. The priority assignment in the interrupt system is problem dependent and is shown in Figure 5. for one typical closed loop guided missile system.

The interconnection strategy is adapted to this structure, so that request for each processor is wrapped to the corresponding interrupt level.

AMS system bus uses the non-bus-vectored mode for interrupts. When an interrupt request line is activated the interrupt controller generates an interrupt vector address and transfers it to the processor over the local bus [17].

4. ONE CONCEPT FOR PARALLELIZATION OF MATHEMATICAL MODELS GIVEN BY ODEs IN DIGITAL REAL TIME SIMULATION

4.0. Real time simulation in multiprocessor environment

The abilities of parallel real time simulation predominantly depend on the performance and architecture of parallel multiprocessor system, types of interaction between parallel processors, on the problem under consideration i.e. its inherent level of parallelism, on numerical methods for numerical integration, strategy of task allocation and finally the cooperation between parallel architecture and parallel discrete time model of a original continuous system. But it is important to note that the main contribution to the improvement of efficiency in multiprocessor real time simulation depends on decomposition of model, on a process of discretization and on mapping of given problem onto parallel multiprocessor architecture. This process usually involves several phases that start with decomposition of mathematical model given by algebraic equations (AEs) and ordinary differential equations (ODEs) or by partial differential equations (PDEs). Since these equations are defined over continuous time domain in the second step some kind of discretization or numerical approximations must be employed in order to enable digital implementation. Finally, it is necessary to perform suitable partitioning of derived discrete time model onto multiprocessor environment. Mapping of decomposed and discretized model onto parallel architectures can be performed in a different way.

In dynamic load balancing, discrete time models are allocated to microprocessors at run time. Discrete models automatically migrate from heavily loaded microprocessor to lightly loaded ones. By attaining a well-balanced load, better processor utilization can be achieved and thus higher performance of complete system.

In a static load balancing method, models are allocated to microprocessors after compile time i.e. before start up run time. Such static techniques require fairly accurate predictions of the resource utilization for each model.

For programs with unpredictable run-time resource utilization, dynamic load balancing is more desirable because it allows the system to continuously adapt to rapidly changing run time conditions. As a popular measure of load balancing it is possible to use CPU time utilization, communication time, the number of concurrent microprocessors in active operation, the number of concurrent models etc..

In real time simulation correct prediction of the computational requirements and resource demands for each software module i.e. for each program must be known in advance to enable real time implementation. This estimation can be derived after the process of discretization. On the basis of this information, algorithm for task allocation provides well load balance and real time execution. Therefore the concept of static load balancing in the real time simulation is a natural one. In simulator design, what is primary interest of our research, the objective function must provide real time simulation of related problem with desired accuracy and with minimal number of microprocessors. The system which enables developments of the simulator in this way can be considered as development system for simulator design and realization. The accepted objective function is natural in designing and realization of digital simulator for operators training or hardware in the loop testing.

Real time simulator described in this article hosted on IBM PC/AT has been realized in order to provide the user with such abilities and furthermore to allow generation of real time machine code for target processor based on Intel microprocessor 8086 and arithmetic coprocessor 8087. Through attached peripheral homogenous tightly coupled multiprocessor system based on single board computers, such simulator system provides user with different experimental abilities in operator training or control system design and testing. The software support of such simulator development system provides the abilities of extensive non real time simulation which leads to such decomposition technique, numerical integration and task allocation strategy which guarantees the minimal number of parallel processing units i.e. minimal hardware complexity necessary for realization of a different kind of simulator system. This hardware complexity strongly depends on desired level of accuracy in process of simulation, so that with increasing level of approximation the number of microprocessor can be significantly increased. The substantial influence on

this facts have choices of decomposition techniques and procedures for numerical integration. Therefore special attention will be dedicated to the relationship between hardware complexity on one side and decomposition techniques, numerical methods for discretizations and algorithm for task allocation on the other side.

Now we can define the procedure for real time simulation in multiprocessor environment with following steps:

Step 1. Structural and dynamics decomposition of system corresponds to physical partitioning of problems into a sequence of modules with lower level of complexity.

Step 2. Tuning between numerical methods and sample rates for discretization and mathematical modules according to its nature; linear or nonlinear, time variant or time invariant, stiff or nonstiff, with or without discontinuities, spectral characteristics of input signals and so on.

Step 3. Task allocation process follows physical arrangement of system and in cooperation with numerical methods for discretization determines desired level of granularity in order to achieve equal load balancing between processors, minimal hardware requirements and real time execution.

Presented procedure is not straightforward, it may be iterative one and all these steps must be taken into account very briefly in order to achieve optimal real time simulation of given problem in accordance with the accepted objective function.

4.1. System decomposition

The first step in the above procedure requires decomposition of a mathematical model of complex dynamic system into a number of hierarchical functional modules or blocks of different complexity. With such a modular or block approach a realistic complex problem can be subdivided into a sequence of smaller modules or blocks. By applying this formalism, complex mathematical models can be easily transformed into one block level distributed structure which enables isolation of standard mathematical models and their further efficient processing. Decomposition scheme, which we shall prefer, is heuristic one and is mainly based on physical partitioning of complex system. Now we shell define the parallelism degree of model on level of block diagram,as the maximum number of functional blocks that can be executed at the same time on different processors if necessary in real time execution. For very fast system i.e. for time critical real time simulation, inherent parallelism degree can be further increased by partitioning from block level to an equation level or even an arithmetic operation level. If it is necessary, it is possible to determine computation of the longest execution time i.e. critical data flow trajectory in running through the block diagram. The efficiency of modular partitioning is problem dependent, but only with such an approach it is possible to perform the optimal adaptation of numerical methods of discretization to each module of distributed system. Such decomposition of a complex system into multiple low level computational modules which enables adaptation of numerical method and period of discretization to each module is most important for digital real time simulation. After structural decomposition which corresponds to physical topology of the system, suitable dynamic decomposition is necessary in order to adapt periods of discretization or integration to each module. The concept of multirate sampling enables different sampling rates in different modules or in different loops, and leads to the significant reduction of computational load i.e. CPU time savings and to improving the numerical conditioning. Such structural and dynamic decomposition detects the level of parallelism which is very often inherent in complex original continuous system. Separation of differential equations according to their type, particularly linear differential equations from nonlinear e.g. nominal trajectory from perturbated, separation according to the spectral characteristic i.e. separation of fast portions from slow ones and separation according to the frequency contents of input signal are substantial for the achievement of high efficient digital real time simulation.

A heuristic method for system decomposition used in this concept is based on the fact that the modules that follow from partitioning of the overall system are very similar to the physical topology of the system. The main advantage of this approach is that the processing blocks are associated with physical sections, and model implementation and verification can be easily done. In addition, variables used in the interprocessor communication have physical meaning, which can aid in the understanding and interpretation of the model. Furthermore by exchanging only the output variables between blocks it is possible to reduce significantly communication requirements between processors.

Because decomposed system is similar to the physical system, this method is problem dependent. However, the advantages of the method far outweigh this disadvantage. The main advantages of this method are:

- Decomposition is easy to make, since it follows the physical arrangement of the system.
- Highly modular approach is very flexible in the case of structural or modules modifications.
- Interprocessor communication requirements are minimized.
- Program design is simplified, which is a direct consequence of modular structure.
- Program coding is straightforward. It is easier

to code the small modules that result from the decomposition than complex ones.

- Checking, testing and debugging is also easier, especially message interchange between microprocessors during interprocessor communications.

This decomposition has been applied on original continuous models and its abilities are determined by the nature of the problem. Modules or blocks that are independent enable simultaneous or concurrent calculation and provide parallel decomposition. Modules or blocks that are sequential lead to the cascade decomposition. This level of parallel or sequential computation is predominately determined by the nature of the problem, but in the following steps it will be shown that this inherent level of parallel or sequential properties can be significantly modified by the choices of numerical methods for discretization or integration.

4.2. Numerical integration

This step transforms original continuous mathematical model into equivalent discrete one, that must guarantee desired level of approximation with minimal arithmetic complexity normalized on some common frame time. By this transformation using different techniques of numerical integration it is possible to additionally increase the degree of model parallelism from the inherent one to some desired level which enables real time simulation of related problems on given multiprocessor configuration. It means that the complete parallelism in discrete time model for implementation depends not only on efficiency of functional decomposition i.e. topology of system, but also on numerical method for discretization. The level of parallelization which will be added or extended with the choices of numerical methods determines the final level of granularity of discrete time model for implementation. The level of granularity in equivalent discrete time model can vary in a range from instructions and statement level to program or a group of programs level. This depends on the complexity of mathematical model and its dynamic characteristics i.e. real time constraints and architecture of multiprocessor system that would be used for implementation. For example, if a model of some problem is represented by a sequence of sequential modules or blocks which require sequential computation, by using any of explicit numerical methods for integration, they can be easily transformed to parallel procedures until the desired level of granularity is reached. Assignment of the suitable integration or discretization method to each module or block, and suitable period of discretization require extensive non real time analysis, checking, testing and verification. In the following part of this article we shall focus our attention on numerical methods that can be considered as suitable for digital real time integration of ordinary differential equations and on dependance analysis between the parallelism of simulation models and numerical methods used for discretization or numerical integration.

For a linear or linearized problem given by

$$\dot{x}(t) = A\,x(t) + B\,u(t) \quad , \quad x(0) = x_0 \tag{1}$$

some modification of standard discretization methods can be considered as optimal one.

Modification of standard step invariant method [20,21] is given by the following equations

$$x^*_{k+1} = \Phi(A,T,W)\,x^*_k + \Gamma(A,B,\lambda,\gamma,T,W)\,u_k \tag{2}$$

where the state vector, and system matrices are determined by

$$\begin{aligned} x^*_k &= W^{-1}x_k \\ \Phi &= W^{-1}e^{AT}W \\ \Gamma &= W^{-1}\lambda[A^{-1}(e^{AT}-I) + \gamma T(e^{AT}-I)]B \end{aligned} \tag{3}$$

State space version of T-integrator is given by [22]

$$x^*_{k+1} = A^*(A,L,\Gamma,T,W)x^*_k + B^*_1(A,L,\Gamma,T,B,W)u_{k+1} + B^*_2(A,L,\Gamma,T,B,W)u_k \tag{4}$$

where is

$$\begin{aligned} x^*_k &= W^{-1}x_k \\ A^* &= [I - LT\Gamma A]^{-1}[I + LT(I-\Gamma)A] \\ B^*_1 &= [I - LT\Gamma A]^{-1}\,LT\Gamma B \\ B^*_2 &= [I - LT\Gamma A]^{-1}\,LT(I-\Gamma)B \end{aligned} \tag{5}$$

For general nonlinear problem given by

$$\dot{x}(t) = f(x(t),t,u(t)) \quad , \quad x(0) = x_0 \tag{6}$$

following numerical integration algorithms can be considered as suitable for real time digital simulation [23];

* Single-pass integration algorithm
  - Euler explicit

$$x_{n+1} = x_n + T\,f(x_n,t_n,u_n) \tag{7}$$

  - AB - 2

$$x_{n+1} = x_n + T/2[3f(x_n,t_n,u_n) - f(x_{n-1},t_{n-1},u_{n-1})] \tag{8}$$

  - AB - 3

$$x_{n+1} = x_n + T/12[23f(x_n,t_n,u_n) - 16f(x_{n-1},t_{n-1},u_{n-1})] + 5f(x_{n-2},t_{n-2},u_{n-2})] \tag{9}$$

* Real time Runge-Kutta version
  - RK - 2

$$\begin{aligned} x^P_{n+1/2} &= x_n + T/2\,f(x_n,t_n,u_n) \\ x_{n+1} &= x_n + T\,f(x^P_{n+1/2},t_{n+1/2},u_{n+1/2}) \end{aligned} \tag{10}$$

  - RK - 3

$$\begin{aligned} x^P_{n+1/3} &= x_n + T/3\,f(x_n,t_n,u_n) \\ x^P_{n+2/3} &= x_n + 2T/3\,f(x^P_{n+1/3},t_{n+1/3},u_{n+1/3}) \\ x_{n+1} &= x_n + T/4[f(x_n,t_n,u_n) + 3f(x^P_{n+2/3},t_{n+2/3},u_{n+2/3})] \end{aligned} \tag{11}$$

* Adams-Moulton serial predictor-corrector
  - AM - 2

$$\begin{aligned} x^P_{n+1} &= x_n + T/2[3f(x_n,t_n,u_n) - f(x_{n-1},t_{n-1},u_{n-1})] \\ x_{n+1} &= x_n + T/2[f(x^P_{n+1},t_{n+1},u_{n+1}) + f(x_n,t_n,u_n)] \end{aligned} \tag{12}$$

-AM - 3

$$x^P_{n+1} = x_n + T/12[23f(x_n, t_n, u_n) - 16f(x_{n-1}, t_{n-1}, u_{n-1}) + 5f(x_{n-2}, t_{n-2}, u_{n-2})] \quad (13)$$

$$x_{n+1} = x_n + T/12[5f(x^P_{n+1}, t_{n+1}, u_{n+1}) + 8f(x_n, t_n, u_n) - f(x_{n-1}, t_{n-1}, u_{n-1})]$$

For complex nonlinear modules or blocks, which unable further physical decomposition and single processor real time implementation, it is necessary to use some technique for equation segmentation or numerical methods for parallel integration. Standard approach for solving such problem [24] requires partitioning the set of n equations and then allocation of a certain number of the n equations to each of the microprocessors to achieve the speed needed for real time simulation. Each microprocessor would be responsible for performing the function evaluations and integrations associated with its assigned equations. Such subset of equations can operate in parallel, only the values which are necessary in the other equations would be transferred periodically between microprocessors. Since the function evaluations i.e. the computation of derivative would be done in parallel, a good deal of time can be reduced in comparison with a single microprocessor implementation. Just how much time is saved depends on;

- How closely coupled is the system of equations;
- How the equations are allocated to the microprocessors;
- What integration algorithm is used for discretization of each subset of equations;
- What types of communication channels and protocols are available between microprocessors.

Alternative solution to the above standard approach is concerned with utilization of parallel predictor-corrector methods. In this case partitioning is performed in the sense of the algorithm i.e. numerical method but not in the sense of the system of equation as in previous approach. One possible parallel predictor-corrector method has been presented by Miranker and Liniger [24] for system $\dot{y} = f(x,y)$ and is given by

$$y^P_{i+1} = y^C_{i-1} + h/3(8f^P_i - 5f^C_{i-1} + 4f^C_{i-2} - f^C_{i-3})$$
$$y^C_i = y^C_{i-1} + h/24(9f^P_i + 19f^C_{i-1} - 5f^C_{i-2} + f^C_{i-3}) \quad (14)$$

For parallel real time simulation parallel block implicit methods can be also very useful. One version of a fourth-order block presented by Shampine and Watts [24] is given by:

- Predictor equations (15)

$$y^P_{i+1} = 1/3(y^C_{i-2} + y^C_{i-1} + y^C_i) + h/6(3f^C_{i-2} - 4f^C_{i-1} + 13f^C_i)$$
$$y^P_{i+2} = 1/3(y^C_{i-2} + y^C_{i-1} + y^C_i) + h/12(29f^C_{i-2} - 72f^C_{i-1} + 79f^C_i)$$

- Corrector equations

$$y^C_{i+1} = y^C_i + h/12(5f^C_i + 8f^P_{i+1} - f^P_{i+2})$$
$$y^C_{i+2} = y^C_i + h/3(f^C_i + 4f^P_{i+1} + f^P_{i+2}) \quad (16)$$

The choice of the optimal discretization method i.e. tuning the method to each separate module requires well understanding of the character and dynamic of each module and well understanding of numerical characteristics of all of the above mentioned numerical methods. But this can be done only on the basis of separation between linear differential equations, and nonlinear, fast portions of the problem from slow portions, time invariant from time variant. It means that the decomposition of problem has great influence on the efficiency of the complete procedure. By such approach we can significantly increase the solution bandwidth of the problem, which is the most important in real time simulation.

4.3. Task allocation

The last step in the presented procedure requires distribution of derived discretized modules among microprocessors in multiprocessor system and can be used in iterative fashion with previous two steps. This process of a task allocation i.e.process of assigning software modules which constitute distributed discretized mathematical model of original continuous system to each processing element, requires an understanding of data or block dependencies that exist among problem variables. There are two different ways in task allocation strategy which depend on applications.

In the first case, architecture of the multiprocessor system is completely defined and determined by the type and number of available processors, their performance and way of their communication. The computing tasks involved in solution of simulation problem, in this case, must be partitioned on the available processors in order to minimize idle time of each processor and to minimize the time lost in the communication of program segments. Furthermore special constraints and limitations on real time processing are not supposed i.e. solution can be generated faster than real time or slower then real time, which depends on the problem and performance of multiprocessor system. In such environment it is necessary to attain such load balancing which leads to a better multiprocessor resource utilization. Well-balanced load provides a higher performance of complete system i.e. minimal total run time. Measures of load balance in this case may include the CPU time utilization for each processor, communication time in relation to the computation time, the number of concurrent processes or modules in simultaneous processing and so on. The combination of these different objectives can be expressed by the fast improvement achieved on multiprocessor system. This improvement can be characterized by factor S, which is ratio between solution times using one processor and N processors[27].

$$S = T_S/T_M \quad (17)$$

where is: T_S - single processor solution time,

T_M - multiprocessor solution time.

The efficiency of complete multiprocessor implementation can be defined by:

$$E = S/N \quad (18)$$

where **N** denotes the number of microprocessors, and **S** is given by (17). Efficiency is expressed as a percentage by multiplying the above expression by 100. The assignment must be made so as to optimize relation (17) and (18), i.e. to enable minimization of execution time.

The second approach to the problem of task allocation is related to design and development of digital real time simulator for operator training or hardware in the loop testing i.e. restricted on real time simulation. Therefore in such applications it may be assumed that the architecture of multiprocessor system is not given in advance and must be determined by this procedure. In this context, task allocation strategy on the basis of results generated in first two steps must provide high fidelity real time simulation and minimization of hardware requirements, since for some kind of simulator system such computer configuration must be duplicated in high number of copies. As already pointed out, in this system all program segments are known in advance as its time required for their execution and therefore static allocation concept will be appropriate one. If the execution times for each module and communication times between processors are known, then the problem of task allocation is to determine such distribution of modules on parallel processors which support real time simulation with minimal hardware requirements. The number of software modules which form one task which will be assigned to one microprocessor depends on arithmetic complexity of problem modules and real time requirements. Possible assignment can be several modules to one microprocessor, or several microprocessors to one module. This relationship depends predominately on the processing power of each processing element, complexity of discretized modules and their dynamics, intensity of interprocessor communication and so on. At the beginning of the procedure we start with allocation of task to first microprocessor on the basis of block level critical path method [28]. For some common frame time we add modules to first microprocessor until the sum of periods of implementation of these modules is less than or equal to this common frame time T. After that we are going on, with assignment of a tasks to the following microprocessors. So that for each microprocessor must be valid

$$\frac{\sum_{i=1}^{n} T_{imp}^{i}}{T_{disc}} \leq 1 \quad (19)$$

where T_{imp}^{i} denotes the period of implementation module i normalized on common frame time **T** and **n** denotes the number of modules assigned to each microprocessor. In this way it is necessary to continue with the process of task allocation for each succeeding microprocessor, until all modules have been allocated. If equation (19) is valid for all modules, single processor implementation of related problems is possible. With such an approach it is possible to reduce idle time of each microprocessor in multiprocessor configuration. But on the other side this approach increases time delay due to finite computation time. This delay may cause the effects of instabilities in closed loop simulation with included external hardware or in the operator training applications.

In the case of fast and complex modules it is very often necessary to assign several microprocessors to one module. Using the equation segmentation methods or some form of parallel predictor corrector algorithms or block implicit methods it is possible to achieve the speed of real time processing.

The level of granularity in the process of task allocation depends predominantly on the architecture of multiprocessor system, decomposition techniques, complexity of modules, their dynamic real time constraints, numerical methods used for discretization and so on. On one side very fast modules lead to the fine level of granularity, for example statement level or instruction level. Massively parallel processor in order to achieve the high efficient utilization of available processing power needs fine level of granularity. On the other side relatively slow modules lead to the high level of granularity preferred in single bus multiprocessor configuration. Different level of granularity leads to the different level of intensity in interprocessor communication. Communication time is not negligible specially in fine level of granulations since communication time is comparable with computation time. In such circumstances, task allocation optimized only on maximization of parallel operation would not be at all optimal one, when communication times are taken into account, especially if a large amount of data is to be a shared. Therefore parallel multiprocessor simulation requires detail consideration and analysis in order to reduce i.e. to minimize interprocessor communication. Waiting for intermediate results and delays during communication time decreases throughput per processor as the number of processors grows.

With careful decomposition schemes which follow physical topology of problems and with choices of suitable methods for numerical discretization and task allocation strategy, hardware requirements for real time simulation can be significantly reduced. It is therefore desirable to perform extensive non real time simulation and analysis in order to determine the best manner of allocating program modules and to achieve the

most cost effective solution.

4.4. Simulator design for one spinning missile system

The above presented procedure can be illustrated by the following example which requires simulator design that must provide real time simulation of one spinning missile according to the desired level of accuracy.

After linearization of 6-DOF model about corresponding nominal trajectory and suitable physical partitioning, block diagram form of one spinning missile can be shown by Figure 6.

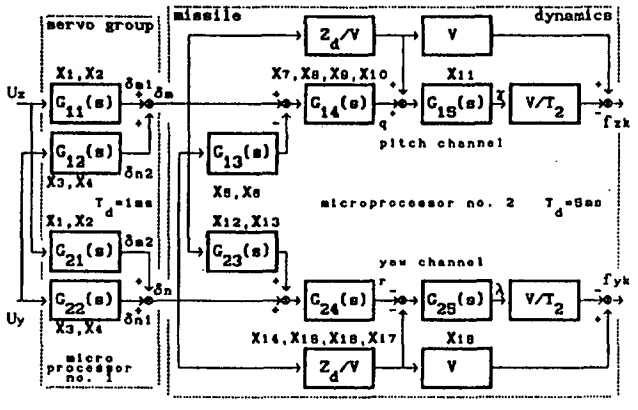

Figure 6. Block diagram of spinning missile in nonrotating coordinate system

Servo group mathematical model can be defined by the following system of ordinary differential and algebraic equations [29],

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -1/T_p^2 \, x_1 - 2\xi/T_p \, x_2 + 1/T_p^2 \, U_z \\
\delta_{m1} &= K_p C_2 \, x_1 + K_p C_1 \, x_2 \\
\delta_{m2} &= K_p D_2 \, x_3 + K_p D_1 \, x_4 \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= -1/T_p^2 \, x_3 - 2\xi/T_p \, x_4 + 1/T_p^2 \, U_y \\
\delta_{n1} &= K_p C_1 \, x_3 + K_p C_2 \, x_4 \\
\delta_{n2} &= K_p D_1 \, x_3 + K_p D_2 \, x_4
\end{aligned} \tag{20}$$

or in state space form with,

$$\begin{aligned}
\dot{x}_{pk} &= A_{pk} \, x_{pk} + B_{pk} \, U_{pk} \\
y_{pk} &= C_{pk} \, x_{pk}
\end{aligned} \tag{21}$$

In this case state, input and output vectors correspond to

$$x_{pk} = [x_1 \; x_2 \; x_3 \; x_4]^T, \quad U_{pk} = [U_z \; U_y]^T \tag{22}$$
$$y_{pk} = [\delta_m \; \delta_n]^T ,$$

while system matrices A_{pk}, B_{pk} and C_{pk} follow directly from (20).

Dynamics equations for pitch channel are given by

$$\begin{aligned}
\dot{x}_5 &= x_6 \\
\dot{x}_6 &= -1/T_0^2 \, x_5 - 2\xi/T_0 \, x_6 + 1/T_0^2 \, \delta_z \\
\delta_{mn} &= K_\sigma \, x_5 + K_\delta T_2 \, x_6 \\
\dot{x}_7 &= x_8 \\
\dot{x}_8 &= x_9 \\
\dot{x}_9 &= x_{10} \\
\dot{x}_{10} &= -a_1 \, x_7 - a_2 \, x_8 - a_3 \, x_9 - a_4 \, x_{10} + a_5(\delta_m - \delta_{mn}) \\
q &= K_q \, x_7 + K_q p_1 \, x_8 + K_q p_2 \, x_9 + K_q p_3 \, x_{10} \\
\dot{x}_{11} &= -1/T_2 \, x_{11} + (q + Z_\delta/V \, \delta_m) \\
\gamma &= x_{11} \\
f_{zk} &= V/T_2 \, \gamma
\end{aligned}$$

State space form of equation (23) is determined by

$$\begin{aligned}
x_{dz} &= [x_5 \; x_6 \; x_7 \; \cdots \; x_{11}]^T \\
u_{dz} &= [\delta_m \; \delta_n]^T \\
\dot{x}_{dz} &= A_{dz} \, x_{dz} + B_{dz} \, U_{dz} \\
f_{zk} &= C_{dz} \, x_{dz} + D_{dz} \, U_{dz}
\end{aligned} \tag{24}$$

Dynamic of yaw channel is identical and is given by,

$$\begin{aligned}
x_{dy} &= [x_{12} \; x_{13} \; x_{14} \; \cdots \; x_{18}]^T \\
u_{dy} &= [\delta_m \; \delta_n]^T \\
\dot{x}_{dy} &= A_{dy} \, x_{dy} + B_{dy} \, U_{dy} \\
f_{yk} &= C_{dy} \, x_{dy} + D_{dy} \, U_{dy}
\end{aligned} \tag{25}$$

Non real time simulation of the above equations has been performed on Cyber 170/850 using standard routine for numerical integration of ODEs from IMSL library (DVERK) in order to generate reference solution. With common period of integration of 1ms, and with 6-order Runge-Kutta methods reference solutions have been obtained and are shown in Figure 7.

Derivation of discrete time model for real time simulation requires careful choice of a method for numerical integration and period of discretization. For this linearized problem one of suitable numerical methods for discretization is the approach given by equation (4) and (5). After detailed dynamic analysis of original continuous system based on eigenvalues inspection and their distribution, the concept of multi rate processing has been accepted. Real time processing in servo group has been performed with 1ms period. In simulation of rigid body dynamic equations the period of discretization is 5ms. The complete discrete time model has been tested through extensive non real time simulation which are shown in Figure 7.

Program for testing discrete time model is based on equations (21),(24),(25) and (4) is given by

```
for i=1 to ....
*servo block
 for j=1 to 5
 input u_pk,k
 x_pk,k = A*_pk x_pk,k-1 + B°_pk (u_pk,k + u'_pk,k-1)
 y_pk,k = C*_pk x_pk,k
 output  y_pk,k(δ_m,k, δ_n,k)
 x_pk,k-1 = x_pk,k
 u_pk,k-1 = u_pk,k
 next j
*dynamic of pitch and yaw channel
 x_dz,k = A*_dz x_dz,k-1 + B°_dz (y_pk,k + y_pk,k-1)
 x_dy,k = A°_dy x_dy,k-1 + B°_dy (y_pk,k + y_pk,k-1)
```

$f_{zk,k} = C^{*}_{dz}\, x_{dz,k} + D^{*}_{dz}\, y_{pk,k}$

$f_{yk,k} = C^{*}_{dy}\, x_{dy,k} + D^{*}_{dy}\, y_{pk,k}$

output f_{zk}, f_{yk}

$x_{dz,k-1} = x_{dz,k}$

$x_{dy,k-1} = x_{dy,k}$

$y_{pk,k-1} = y_{pk,k}$

next i

Comparative analysis of results presented in Figure 7, shows a good tuning of discrete time model that has been provided with compensation matrices L=I and Γ=1/2I (bilinear transformation) and corresponding periods of discretization.

Strategy of static task allocation which follows physical decomposition of problem and choices of the numerical methods for discretization must meet requirements for real time simulation with minimal hardware requirements. Computational load i.e. speed up factor of first module-servo group is determined by period of discretization (1ms) divided by execution time of arithmetic operations needed for implementation of this discrete time model. The speed up factor computed for this module gives K_{servo}=1.0013. This means that real time simulation of this module is possible on only one microprocessor board and that the load balancing for this microprocessor is near ideal. The speed up factor for the following group which presents dynamics of pitch and yaw channel is $K_{dinamic}$=1.00124. This means that real time simulation

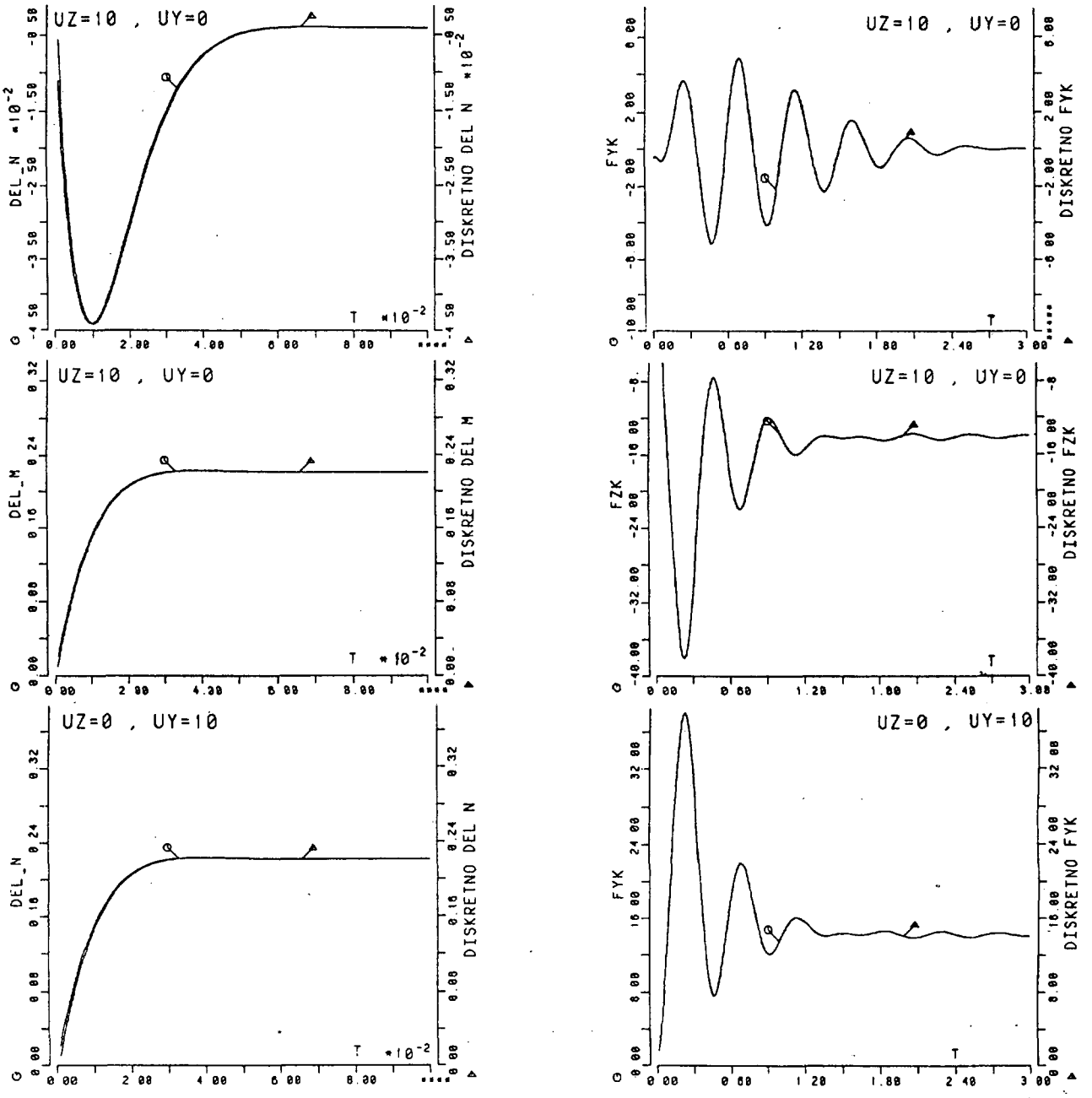

Figure 7. Time responses of continuous and equivalent discrete time models
(T_{servo}= 1ms, T_{dynam}= 5ms, L = I, Γ = 1/2 I)

of this module is also, possible and that it also needs only one microprocessor with near ideal load balancing. From the obtained results follows that the real time simulation of the model given in Figure 6, is possible with two near ideal equally load balance microprocessors. Assignment of tasks is determined by allocation of the discrete time model of servo group to microprocessor no.1, while pitch and yaw dynamic equation to microprocessor no.2. The intermediate results computed by first microprocessor must be transferred to the second one. However second microprocessor must check that intermediate results or sequence of computation and require synchronization between microprocessor no. 1 and microprocessor no. 2. To insure correct synchronization and acceptance of correct results, source microprocessor also broadcasts a ready flag with its results. The destination microprocessor i.e. the second microprocessor waits for ready flag before using the result broadcasted by the first microprocessor. Than it resets the ready flag after it detects that ready flag is set. Procedure executed by a source and destination microprocessor in transferring and waiting for exchange data and variables are standard in such cases.

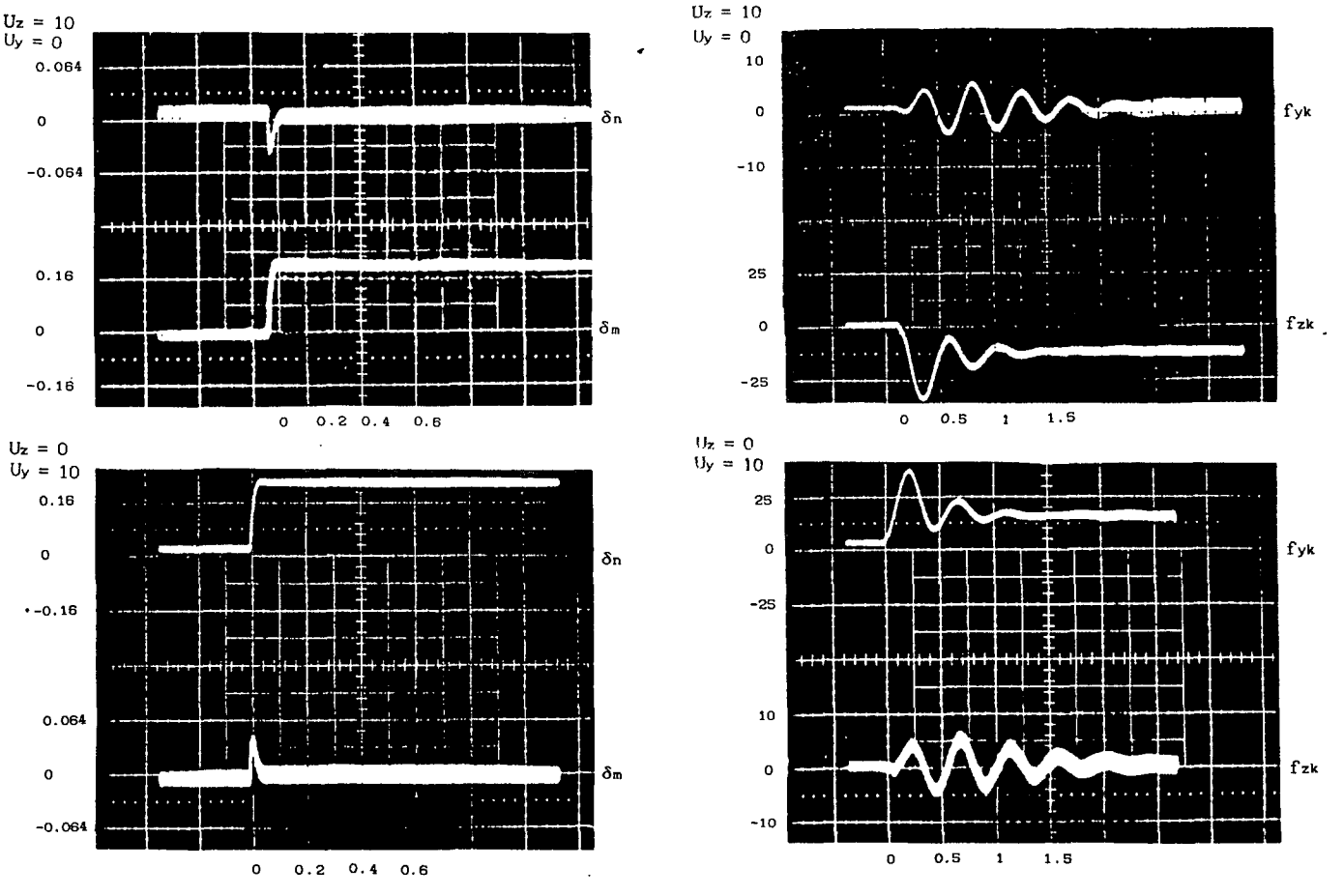

Figure 8. Real time simulation of one spinning missile system

data have been received before using them. In this example we have used for integration implicit method from accuracy and stability reason. But implicit integration methods are not suitable for parallel simulation, since they require careful consideration to insure correct synchronization between microprocessors. In execution of iteration k, first microprocessor must compute the output of servo block $y_{pk,k}$ and broadcast the results to second microprocessor as soon as possible. The second microprocessor in corresponding iteration for computation of its state vector and output equation must wait for values $y_{pk,k}$ from first microprocessor. These data dependencies that exist among servo block and dynamic block determine the

Program for parallel real time simulation for these two microprocessors is coded in assembly language according to the following relations.

```
*microprocessor no. 1 - servo block
*
*interrupt routine (real time clock - 8253 - 1ms)
*
 input u_pk,k          (from a/d converter over AMS-bus)
 x_pk,k = x^P_pk,k + B*_pk u_pk,k
 y_pk,k = C*_pk x_pk,k
 output  y_pk,k              (to microprocessor no.2
                              over dual port memory)
 x^P_pk,k = A*_pk x_pk,k + B*_pk u_pk,k
 idle                  (wait for next real time clock)
```

```
*
*microprocessor no.2 -dynamic of pitch and
*yaw channel
*
*interrupt routine (real time clock - 8253 - 5ms)
input y              (from microprocessor no.1 over
                     dual port memory - wait for
                     semaphor synchronization)
x_dz,k = x^P_dz,k + B*_dz y_pk,k
x_dy,k = x^P_dy,k + B*_dy y_pk,k
f_zk,k = C*_dz x_dz,k + D*_dz y_pk,k
f_yk,k = C*_dy x_dy,k + D*_dy y_pk,k
output f_zk,k, f_yk,k   (to d/a converters over
                         AMS - bus)
x^P_dz,k = A*_dz x_dz,k + B*_dz y_pk,k
x_dy,k = A*_dy x_dy,k + B*_dy y_pk,k
idle                 (wait for next real time clock)
```

The obtained real time solutions that satisfy the accepted objective function are shown in Figure 8 and can be considered as near optimal ones.

5. CONCLUSION

Presented methodology enables high efficient real time integration of complex dynamic system described by ordinary differential equations on multiprocessor system. It means that in the simulator design it is the most important to find the optimal combination of decomposition techniques, discretization algorithms and strategy of task allocation, that leads to the minimal number of microprocessors necessary for simulator realization in agreement with desired accuracy specifications. Through the attached peripheral homogenous single bus tightly coupled multiprocessor system, realized digital simulator provides a user with different experimental abilities in control system design, testing, modification and in the operator training.

6. REFERENCES

[1] W. H. Christal, D. G. Mackey: Guidance and Control Simulations for Laser Guided Weapons, Proceedings of the Conference on Aerospace Simulation, SCS, February 1984.

[2] K. Ćosić, S. Deskovski: Digitalna simulacija prostornog kretanja rotirajuće rakete u realnom vremenu, XXVIII konferencija ETAN-a, Split, 1984.

[3] K. L. Hall: A simulation system architecture for real time applications, The Proceedings of the Summer Computer Simulation Conference, 1986.

[4] K. Ćosić, S. Deskovski, I. Miler, M. Slamić : Digitalni simulator za razvoj i testiranje PO sistema vodenja, XXXI konferencija ETAN-a, Bled, 1987.

[5] J. W. Karplus: Computer hardware in the simulation, Lecture notes, ETH Zürich 1986.

[6] R. J. Hickman: Hardware-in-the-loop simulation of dynamics system with high performance multi -purpose digital computers, The Proceedings of the Summer Computer Simulation Conference, 1987.

[7] A. C. Watts: Aerospace system simulation at Sandia National Laboratories, The Proceedings of the Summer Computer Simulation Conference, 1987.

[8] R. A. Bleck, D. J. Arpasi: Hardware for a real-time multiprocessor simulator, The Proceedings of the Summer Computer Simulation Conference, 1985.

[9] R. Gluck: Hope for simulating flexible spacecraft, Aerospace America, November 1986.

[10] J. O. Hamblen: Parallel continuous system simulation using the Transputer, Simulation, December 1987.

[11] K. Owen: Rise of the supercomputer, Aerospace America, July 1988.

[12] D. Ghosal, L. M. Patniak: SHAMP: An Experimental Shared Memory Multiprocessor System for Performance Evaluation of Parallel Algorithms, Microprocessing and Microprogramming 19, 1987.

[13] E. Pearse O'Grady, C. H. Wang: Multibus-based parallel processor for simulation, Proceedings of the Conference on Aerospace Simulation, SCS, February 1983.

[14] K. Ćosić, I. Miler, D. Hadžiomerović: Koncept multiprocesorskog simulatora za testiranje sistema vodenja u realnom vremenu, Zbornik MIPRO 88, 1988.

[15] K. Ćosić, S. Deskovski, I. Miler, M. Slamić, I. Kopriva: Razvoj multiprocesorskog ekspertnog sistema za projektiranje sistema vodenja i upravljanja, XXXII konferencija ETAN-a, 1988.

[16] M. C. Gilliland, B. J. Smith, W. Calvert: HEP: A semaphore - synchronized multiprocessor with central control, Proceedings of the Conference on Aerospace Simulation, SCS, February 1976.

[17] H. Kirrmann: Events and Interrupts in Tightly Coupled Multiprocessors, IEEE Micro, February 1985.

[18] K. Ćosić, S. Deskovski: PC-based development system for simulator design, 3rd European Simulation Congress, Edinburgh 1989 (to be published).

[19] J. R. Pimentel; Real time simulation using multiple microcomputers, Simulation, March 1983.

[20] K. Ćosić, I. Kopriva: Sinteza diskretnih modela za digitalne simulacije u realnom vremenu, Automatika, br.5-6, 1987.

[21] K. Ćosić, I. Kopriva: Design of the optimal discrete time model for digital real time simulation, European Simulation Multiconference, Roma 1989.

[22] K. Ćosić: Design and implementation of discrete time model for real time digital simulation, All About Simulators, Simulators series, vol. 14, No.1, 1984.

[23] R. M. Howe: Special considerations in real time digital simulation, Summer Computer Simulation Conference, 1983.

[24] M. Franklin: Parallel solution of ordinary differential equations, IEEE Transactions on Computers, vol.c-27, No.5, May 1978.

[25] O. A. Palusinski: Simulation of dynamic systems using partitioning and multirate integration techniques, Summer Computer Simulation Conference, 1983.

[26] O. A. Palusinski: Simulation methods for combined linear and nonlinear systems, Simulation, March 1978.

[27] E. Pearse O'Grady, Chang-Hsein Wang; Parallel processor performance in a jet engine simulation, Summer Computer Simulation Conference, 1984.

[28] A. Makol, W. J. Karplus: ALI: A CSSL/multiprocessor software interface, Simulation, August 1987.

[29] S. Deskovski, M. Slamić: Matematički modeli i simulacije sistema upravljanja rotirajućih raketa, Naučno tehnički pregled, vol.XXXIV, br.1., 1984.