 CMES, vol.107, no.6, pp.447-462, 2015

The Accuracy of Mathematical Models in Simulator Distributed Computing

I. Kvasnica[1], **P. Kvasnica**[2]

Abstract: The issue of simulation of decentralized mathematical models is discussed in the paper. The authors' knowledge is based on a theory of design of decentralized computer control systems. Their knowledge is gained in the process of designing mathematical models that are simulated. A decomposed control system is required to meet the conditions of observation and control. The methodology of a multi-model design is based on main principles of object orientation such as abstraction, hierarchy, and modularity. Modelling on a parallel architecture has an impact on a simulator system. The system is defined by the equations shown below. An important part is the way of analyzing the simulation method, an analytical approach, and corresponding software implementation tools.

Keywords: Accuracy of integration; Centralized and decentralized systems; Control modules; Distributed computing; Modelling and simulation.

1 Introduction

A distributed mathematical model can be simulated using parallel computer architecture which can be based on multiprocessors; each processor is of multi-core architecture. Computational power of such systems is higher compared to single-processor systems, see Duncan, Gordon, Zaluska, and Edwards (1994).

In a process of modelling and simulation of complex systems, various methods of model design and simulation experiment control can be used. It turns out to be very important so that several different selected methods and simulation tools could be combined in a single model, Kvasnica, Páleník, and Čižmár (2007).

The most important concept is the concept of encapsulation allowing hiding of

[1] Regional Department for Environmental Issues of Trencin, Hviezdoslavova 3, 911 00 Trenčín, Slovak Republic. tel: +421-32-7432032, e-mail : kvasnica.igor@gmail.com.

[2] Alexander Dubcek University of Trencin, Faculty of Defense Technology, Department of Informatics, Študentská 2, 911 50 Trenčín, Slovak Republic, +421-32-7400704, Fax: +421-32-7400102, Email: peter1.kvasnica@gmail.com.

implementation details of model parts (sub-models). It is then possible to change the underlying implementation of sub-models easily. We use standard terms to describe the model creation and simulation process, see Hrubý, Kočí, Peringer, and Rábová (2002):

- A *model* defines the structure and functionality of a system;
- An *environment* of a model is connected to an input/output interface of the model;
- A *simulator* provides a background for model behaviour implementation;
- An *experiment* controls the process of simulation via the model;
- A *computing environment* is a tool for implementation of a model and experiments by it.

We have defined the following categories of simulation abstractions:

- Model abstractions;
- Environmental abstractions;
- Abstractions for describing experiments.

We use a continuous simulation language to solve differential equations.

2 Description of mathematical models

Two most basic algorithms, Forward Euler and Backward Euler, are used in our solution for numerical integration. A numerical stability domain is introduced as a pillar to characterize an integration algorithm and a general procedure to find the numerical stability domain of any integration scheme, see Cellier and Kofman (2006).

2.1 Principles of numerical integration

A state-space model is as follows:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \tag{1}$$

Where $\boldsymbol{x}$ is a state vector, $\boldsymbol{u}$ is an input vector, and t represents time, with a set of initial conditions:

$$\boldsymbol{x}_{(t=t_0)} = \boldsymbol{x}_0. \tag{2}$$

Let $x_i(t)$ represent the i^{th} state trajectory as a function of simulated time t. As long as the state-space model does not contain any discontinuity in either $f_i(\boldsymbol{x},\boldsymbol{u},t)$ or any of higher derivatives, $x_i(t)$ is itself a continuous function of time. Such function can be approximated by any desired precision by a Taylor-Series expansion of any given point along its trajectory. As long as the function does not exhibit a finite escape time, i.e. it approaches infinity for any finite value of time. Let t^* denote a point in time, about which we wish to approximate the trajectory using Taylor Series, and let t^*+h be the point in time, at which we wish to evaluate the approximation, Cellier and Kofman (2006). The value of the trajectory at that point can then be given as follows:

$$x_i(t^*+h) = x_i(t^*) + \frac{dx_i(t^*)}{dt}\cdot h + \frac{d^2x_i(t^*)}{dt^2}\cdot\frac{h^2}{2!} + \cdots . \tag{3}$$

Different integration algorithms vary in how they approximate the higher state derivates, and in the number of terms of Taylor-Series expansion, that they consider in the approximation [Rolf and Staples (1986)].

2.2 The approximation accuracy

If the term $n+1$ of the Taylor-Series is considered, the approximation accuracy of the second derivative $d^2x_i(t^*)/d^2t = df_i(t^*)/dt$ should be of order $n-2$, since this factor is multiplied by h^2. The accuracy of the third state derivates should be of the order $n-3$, since this factor is multiplied by h^3, etc. In this way, the approximation is correct up to h^n. N is therefore called the *approximation order* of the integration method; see Cellier and Kofman (2006).

Many engineering simulation applications require a global relative accuracy of approximately 0.002. If the *local integration* error is of size e_l, then the *per-unit-step integration error* assumes the value of $e_{p.u.s} = e_l/h$. The *global integration error* is proportional to the per-unit-step integration error, as long as the integration error does not accumulate excessively across multiple steps.

In accordance with the previously made observation, this corresponds to an algorithm with an approximation order of h^4 for the local integration error. We should require for example a local accuracy of 0.0001 [Rolf and Staples (1986)].

In a digital computer, a real number can only be represented with a finite precision. This type of error is called *round-off error*. It occurs in one of the two main general formats that have become common and called *floating point.*

In common programming languages, there are two formats called single precision for 32-bit numbers and double precision for 64-bit numbers. The most common problems resulting from round-off error occur when many steps are involved with rounding occurring at each step.

3 An appropriate mathematical model of a simulator system

For notation of mathematical models of a flying simulator, we can use the state space description. We have a linear controllable non-observed dynamic system, see Clark (1996):

$$\begin{aligned}
&\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t),\\
&\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t),\\
&\boldsymbol{x}(t) = \boldsymbol{x}_0,
\end{aligned} \tag{4}$$

Where: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{x}$, $\boldsymbol{u}$, and $\boldsymbol{y}$ have dimensions $(n\times n),(r\times n),(l\times n),(n\times l),(m\times l)$, and $(r\times l)$ matrices respectively. The first number in brackets means a number of matrix rows; second number means a number of matrix columns. When we try and make the task easier that we will focus on the object of the control, the equation (1) can have a general shape:

$$\begin{aligned}
&\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u},\\
&\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}.
\end{aligned} \tag{5}$$

The Eq. 5 represent, in a form of a matrix, an aircraft dynamic system of a flying simulator comprising of 11 state variable sensors of information, 18 state variables that express situation coordinates of performing elements in the system. They are divided into two halves and the rest is divided into 38 state variables that represent unmeasured noise and sensor failures [Bajborodin (1975)]. Four parts of a piloting control system are expressed by the state vector $n = 4$ that represent a state matrix.

3.1 A decentralized mathematical model of an aircraft in a simulator

According to the given facts, the first equation from Eq. 5 can be expressed [Lazar, Adamčík, and Labún (2007)]:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}. \tag{6}$$

Let us decompose the given system into four subsystems. The first one shall be the subsystem of state variable sensors, the second and the third ones shall be subsystems of performing items (two systems), noise and failures of the apparatus shall be measured by the fourth one. Thus the subsystems have the order $n_1 = 1$, $n_2 = 1$,

$n_3 = 1$, $n_4 = 1$, of course it does not have to be like this. The state space is divided into 4 parts:

$$\boldsymbol{x} = (x_1, x_2, x_3, x_4)^T = (x_1x_1, x_1x_2, x_1x_3, \ldots, x_4x_4), \tag{7}$$

Where $\boldsymbol{x}_i\boldsymbol{x}_j$ represent the items of a state vector. If i represents order relevancy n, i.e. the number of the subsystem, then j – stands for sequential number of the item in the given subsystem. The architecture of the system matrix $\boldsymbol{A}$ in the state space can be formed after multiplying the following shape, see Lazar, Adamčík, and Labún (2007):

$$\begin{aligned}
\dot{x}_{11} &= a_{11}x_1x_1 + a_{11}x_1x_2 + a_{11}x_1x_3 + a_{11}x_1x_4,\\
\dot{x}_{12} &= a_{12}x_1x_1 + a_{12}x_1x_2 + a_{12}x_1x_3 + a_{12}x_1x_4,\\
&\vdots\\
\dot{x}_{44} &= a_{44}x_1x_1 + a_{44}x_1x_2 + a_{44}x_1x_3 + a_{44}x_1x_4,
\end{aligned} \tag{8}$$

Then we decompose 16 subsystems into four parts called isolated subsystems. The above mentioned process of decomposition is carried out by matrix and vector operations. Sixteen blocks $A_{11}, A_{12}, \ldots, A_{43}, A_{44}$ of the matrix $\boldsymbol{A}$ are marked according to the order, where:

$$\begin{aligned}
\dot{x}_{11} &= A_{11}x_{11} + A_{11}x_{12} + A_{11}x_{13} + A_{11}x_{14},\\
\dot{x}_{12} &= A_{12}x_{11} + A_{12}x_{12} + A_{12}x_{13} + A_{12}x_{14},\\
&\vdots\\
\dot{x}_{44} &= A_{44}x_{11} + A_{44}x_{12} + A_{44}x_{13} + A_{44}x_{14}.
\end{aligned} \tag{9}$$

The mathematical description of isolated subsystems has then the following form:

$$\dot{x}_1 = A'_{11}x_1, \dot{x}_2 = A'_{22}x_2, \dot{x}_3 = A'_{33}x_3, \dot{x}_4 = A'_{44}x_4, \tag{10}$$

Where:

$$A'_{11} = \begin{pmatrix} A_{11}\\ A_{12}\\ A_{13}\\ A_{14} \end{pmatrix}, A'_{22} = \begin{pmatrix} A_{21}\\ A_{22}\\ A_{23}\\ A_{24} \end{pmatrix}, A'_{33} = \begin{pmatrix} A_{31}\\ A_{32}\\ A_{33}\\ A_{34} \end{pmatrix}, A'_{44} = \begin{pmatrix} A_{41}\\ A_{42}\\ A_{43}\\ A_{44} \end{pmatrix}. \tag{11}$$

The mutual relations between the first and second isolated subsystems are described by $l_{12}(x)$ meaning that the equation of the first and second isolated subsystem is:

$$l_{12}(x) = A'_{11}\begin{pmatrix} x_1\\ x_2\\ 0\\ 0 \end{pmatrix} = (A_{11}, A_{12}, A_{13}, A_{14})^T \begin{pmatrix} x_1\\ x_2\\ 0\\ 0 \end{pmatrix}. \tag{12}$$

The analyzed case is simple and well described in the reference of Blakelock (1991). The capability of being decomposed can be calculated by means of incidental matrices that can be utilized in cases when the mathematical model of the system is known. Units of incidental matrix are placed when its elements are permuted or transformed so that they can appear diagonally. The number of items on the main diagonal is given by the number of subsystems.

4 A mathematical model of speed and angle of attack in a simulator

The change of parameters of flying objects is described by differential equations in a mathematical incremental form. To create such system of differential equations, one must know aerodynamic coefficients, a mathematical model of aircraft systems and other parameters of aircraft. Mathematical models of a flight simulator in Laplace transformation, see Blakelock (1991), Krasovkij (1980), are created by this approach.

4.1 Speed increment dependence

The speed increment is defined by the equation:

$$\Delta V(s) = -W_V^{\delta_T}(s)\Delta\delta_T(s) - W_V^{\delta_B}(s)\Delta\delta_B(s), \tag{13}$$

Where $W_V^{\delta T}(s)$ is the mathematical model of speed (a transfer function) depending on the fuel supply, $\delta_T(s)$ is the fuel supply, $W_V^{\delta B}(s)$ is a mathematical model of speed depending on the elevator angle, $\delta_B(s)$ is the elevator angle. From the mathematical model of speed in the longitudinal direction, the fuel supply and angle of attack can be determined, see Krasovkij (1980):

$$W_V^{\delta_T}(s) = -a_x^{\delta T}\frac{\Delta_{11}}{\Delta}, \quad W_V^{\delta_B}(s) = -a_y^{\delta B}\frac{\Delta_{21}}{\Delta} - a_{mz}^{\delta B}\frac{\Delta_{31}}{\Delta}, \tag{14}$$

Where $a_x^{\delta T}$ is a speed coefficient with respect to fuel supply, $a_y^{\delta B}$ is a lift coefficient with respect to the elevator angle, $a_{mz}^{\delta B}$ is a speed angle coefficient with respect to the elevator angle, $\Delta(s)$ – a determinant of the transfer function, $\Delta_{11}(s)$, $\Delta_{21}(s)$, $\Delta_{31}(s)$ – algebraic adjuncts to the determinant $\Delta(s)$. The transfer function of fuel supply can be calculated using the following equation:

$$\begin{aligned}\Delta V_V^{\delta_T}(s) &= W_V^{\delta_T}(s) * \Delta\delta_T(s) \\ &= 5\frac{s^3 + 1.12s^2 + 62.782s + 25.32}{s^4 + 1.1338s^3 + 62.7975s^2 + 28.6585s + 4.09291}\Delta\delta_T(s).\end{aligned} \tag{15}$$

The transfer function of the elevator is given by:

$$\begin{aligned}\Delta V_V^{\delta_B}(s) &= W_V^{\delta_B}(s) * \Delta\delta_B(s) \\ &= \frac{-0.11\cdot(9.81s+620.973)-0.42\cdot(-9.81s-10.0062)}{s^4+1.1338s^3+62.7975s^2+28.6585s+4.09291}\Delta\delta_B(s).\end{aligned} \tag{16}$$

Information about mathematical solution of these equations is known [Clark (1996)]. The coefficients for $l_{12}(x)$ defined by the equation (12) are as follows:

$$A_{11} = 5\frac{s^3+1.12s^2+62.782s+25.32}{s^4+1.1338s^3+62.7975s^2+28.6585s+4.09291}, x_1 = \Delta\delta_T(s). \tag{17}$$

$$A_{12} = \frac{-0.11\cdot(9.81s+620.973)-0.42\cdot(-9.81s-10.0062)}{s^4+1.1338s^3+62.7975s^2+28.6585s+4.09291}, x_2 = \Delta\delta_B(s). \tag{18}$$

4.2 Angle of attack dependence

The increment of an angle of attack is defined as follows:

$$\Delta\alpha(s) = -W_\alpha^{\delta_T}(s)\Delta\delta_T(s) - W_\alpha^{\delta_B}(s)\Delta\delta_B(s), \tag{19}$$

Where $W_V^{\delta T}(s)$ is a mathematical model of speed (a transfer function) depending on the fuel supply, $\delta_T(s)$ is the fuel supply, $W_V^{\delta B}(s)$ is a mathematical model of speed depending on the elevator angle, $\delta_B(s)$ is the elevator angle. From the mathematical model of an angle of attack in the longitudinal direction, the fuel supply and the angle of attack can be determined, see Krasovkij (1980):

$$W_\alpha^{\delta_T}(s) = -a_x^{\delta T}\frac{\Delta_{12}}{\Delta}, \quad W_\alpha^{\delta_B}(s) = -a_y^{\delta B}\frac{\Delta_{22}}{\Delta} - a_{mz}^{\delta B}\frac{\Delta_{32}}{\Delta}, \tag{20}$$

The meaning of coefficients $a_x^{\delta T}$, $a_y^{\delta B}$, $a_{mz}^{\delta B}$ and $\Delta_{xx}(s)$ see the paragraph after Eqs. 14. The transfer function of fuel supply can be calculated using the following equation:

$$\begin{aligned}\Delta\alpha_\alpha^{\delta_T}(s) &= W_\alpha^{\delta_T}(s) * \Delta\delta_T(s) \\ &= 5\frac{0.002s^2-0.2518s-0.1}{s^4+1.1338s^3+62.7975s^2+28.6585s+4.09291}\Delta\delta_T(s).\end{aligned} \tag{21}$$

The transfer function of the elevator is as follows:

$$\Delta\alpha_\alpha^{\delta_B}(s) = W_\alpha^{\delta_B}(s).\Delta\delta_B(s) = \frac{-0.11\cdot(-s^3+0.8862s^2+0.012422s-2.4525)-0.42\cdot(-s^2-0.4138s-0.02514)}{s^4+1.1338s^3+62.7975s^2+28.6585s+4.09291}$$

$$\Delta\delta_B(s). \tag{22}$$

Next, we define:

$$A_{21} = 5\frac{0.002s^2 - 0.2518s - 0.1}{s^4 + 1.1338s^3 + 62.7975s^2 + 28.6585s + 4.09291},\ x_1 = \Delta\delta_T(s). \tag{23}$$

$$A_{22} = \frac{-0.11\cdot(-s^3+0.8862s^2+0.012422s-2.4525)-0.42\cdot(-s^2-0.4138s-0.02514)}{s^4+1.1338s^3+62.7975s^2+28.6585s+4.09291},$$

$$x_2 = \Delta\delta_B(s). \tag{24}$$

The Eqs.15 and 21 define the responses to the step change of fuel supply in Laplace transformation $\Delta\delta_T(s) = 1/\text{s}$. Equations 16 and 22 define the responses to the step change of an elevator position in Laplace transformation $\Delta\delta_B(s) = 1/\text{s}$.

The stability of linear control system by Root-locus technique of the characteristic equation is defined. The characteristic equation is the denominator Eq. 15 resp. Eq. 21. If all real parts of the roots of this equation are a negative number, the system is stable. Roots of the characteristic equation are a pair of complex conjugated poles $-0.229 \pm \text{j}0.114$ and a pair of complex conjugated poles $-0.338 \pm \text{j}7.894$.

5 Simulation and visualization of models

Latest computer technologies allow formulation and solutions to a new intricate problem which depends on construction of complex mathematical models and methods of their solution. The designing and building a visual model of aircraft processes at fuel supply of aircraft engines must be accurate. Getting the exact solution to this problem requires simultaneous solution for the whole complex of physical and geometrical problems which is based on significant computing resources, Tereshenko (2009).

The simulation and visualization is done using three threads:

- One thread (light weight process) simulates a mathematical model that is waiting for drawing into a graphical window and it puts the values into a buffer;
- Second thread (light weight process) uses data in the buffer and draws the graphs using the GDI+ technology;
- The synchronizing thread controls the main graphical window and its components.

Sequential run of mathematical model program is characterized by equations computing in single computer time. The code operations are realized sequentially in a given order. A disadvantage of this method is a power constraint of the processor that computes the models, see Chapman, Jost, and Van der Pas (2007).

5.1 Parallel execution support

The parallel run of a program code can be the solution of the abovementioned problem. It can be realized using distributed memory (DM) architecture (MPI Control) or shared memory (SM) architecture (OpenMP Control) and hybrid architecture (Hybrid memory).

These simulation computers of a parallel program in distributed computer systems are identified as node computers [Martincová, Grondžák, and Zábovský (2008)]. They usually consist of a primary input message queue, one or more equivalent processors and required equipment for communication via patch links.

The threads operate in either serial or parallel modes. In the *serial mode,* tasks run sequentially on available resources in the nodes. *Input models* represent control values of mathematical models – the fuel supply or the elevator angle. *Output models* represent simulated values of mathematical models – the speed or the attack angle. The given system can be simulated by more computers or processors. In case more processors of a simulator system are involved $(P_1, P_2, \ldots, P_n)$, they communicate with each other by means of a SM, see Fig. 1.

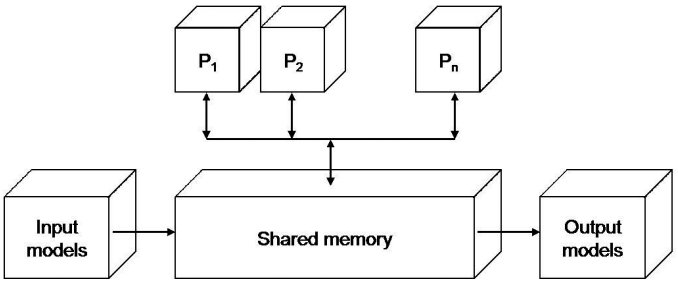

Figure 1: Block diagram of processors in a simulation system.

6 The program application created by MPI

The MPI facilitates this approach by providing many wrappers for calls to industrial-strength open source MPI implementations such as MPICH and LAM-MPI. The applying standard parallel processing techniques and Message Passing Interface (MPI) implementations are used. Their applications can benefit the advantages of parallel computing [Raeth (2010)].

The abovementioned system affects the multiprocessing program code. Distributed architecture was realized as connection of five nodes (one is a central computer, the others are computing nodes), see Fig. 2. One node (N_1) is designed as a central computer and the others are computing ones and each of them is calculating only one mathematical model. As it results from the expression, the mathematical model defined by Eq. 17 $A_{11} * x_1$ is simulated by the N_2 computer on the second node, Eq. 18 $A_{12} * x_2$ is simulated by the N_3 computer on the third node, the mathematical model defined by Eq. 23 $A_{21} * x_1$ is simulated by the N_4 computer on the fourth node resp. Eq. 24 $A_{22} * x_2$ is simulated by the N_5 computer on the fifth node.

Functions send/receive commands are implemented to change messages in the source code application and are added to run in the nodes. We use two basic functions to send and receive messages [Mpich2 (2008)]:

- MPI_Send (parameters)
- MPI_Recv (parameters),

Where the MPI_Send() function on a side of the sender is responsible for sending messages. The corresponding MPI_Recv() function is inserted into a target process to receive messages.

The simulation takes 25 seconds and the intermediate data is sent in periodical time to the node that presents the received data in a graphical form. The simulation time is set depending on the integration error value that is less than 0.002 in distributed simulation methods.

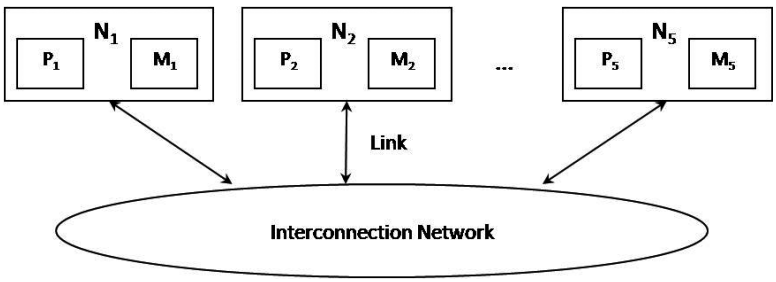

Figure 2: Message passing interface – architecture, N_i – node-computer, P_i – processor, M_i – local memory, Huges (2003).

7 The distributed methods

7.1 Computation using the MPI tool

The parallel system based on the standard MPI can be introduced as the first one. The MPI is a library specification for message passing. Message passing systems

provide alternative methods for communication and movement of data among multiprocessors. Typically it combines local memory and the processor at each node of the interconnection network. There is no global memory, it is necessary to move data from one local memory to another by means of message passing, see Chevance (2005).

There are n nodes which consist of a processor P and a local memory M. Nodes communicate with each other by means of links and via an interconnection network. In executing a given program, the program is divided into concurrent processes, each is executed in a separate processor. This simultaneous execution of the same task in multiple processors is used in order to obtain results faster.

The implementation MPICH2 is a portable, high-performance implementation of the entire MPI-2 standard and consists of a library of routines that can be called from the program. The TOOLKIT is an integrated suite of tools that supports measurement, analysis, attribution, and presentation of application performance for both sequential and parallel programs [Adhianto, Banerjee, Fagan, Krentel, Marin, Mellor-Crummey, and Tallent (2010)].

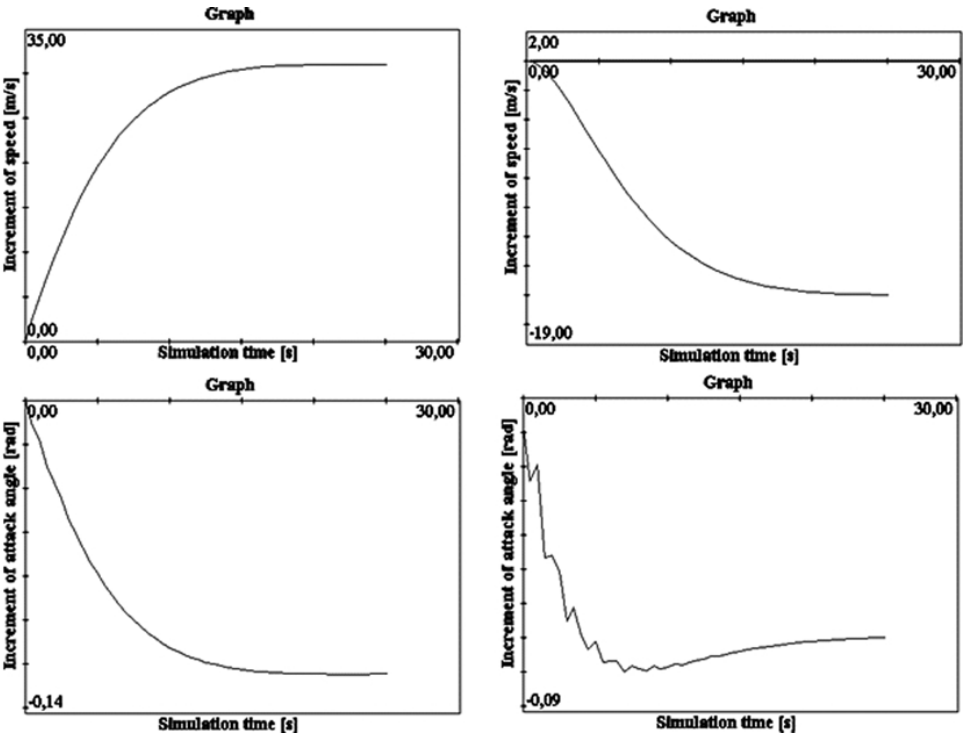

Figure 3: Simulation results of Eq. 17 – upper left, 18 – upper right, 23 – bottom left, 24 – bottom right.

From the graphical output of the central node, we obtain Fig. 3 as a result of the simulation. The simulation of a mathematical model using a cluster technology of a flight simulator is done according to the Eq. 17 and 18 or 23 and 24.

The upper left picture shows the speed increment depending on the fuel supply and it is equal to 31.0192 [m/s]. The upper right picture shows the speed increment depending on the elevator and it is equal to -15.6142 [m/s]. The bottom left picture shows the increment of the angle of attack depending on the fuel supply and it is equal to -0.1237 [rad]. The bottom right picture shows the increment of the angle of attack depending on the elevator and it is equal to -0.0714 [rad].

7.2 Computation using the OpenMP tool

Shared memory computing is based on multi-core processors that can issue multiple instructions per cycle from multiple instruction streams, El-Rewini, Abd-El-Barr (2005). The same simulation problem described above was also realized on the SM based on the OpenMP standard that supports multi-platform SM parallel programming in C/C++.

The presented OpenMP based system is modelled on a computer that consists of Intel Quad Core Q9450 processor with 4 cores, 2.66 GHz each. Informative simulation results are shown in Fig. 4.

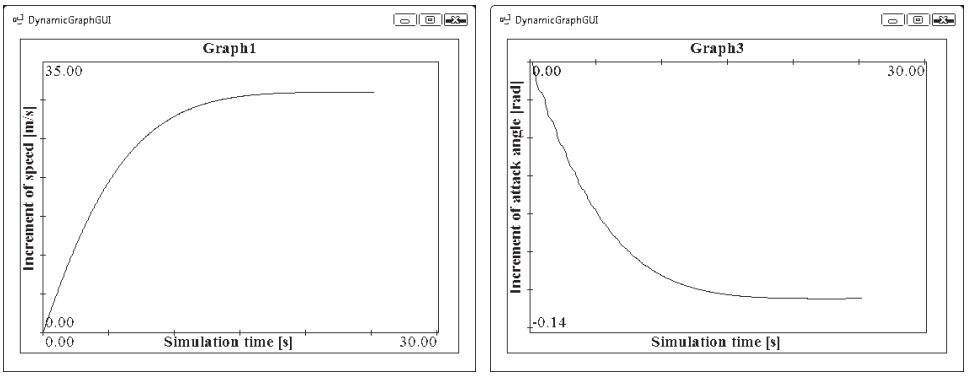

Figure 4: Simulation results of Eq. 17 – left, 23 – right.

The left picture shows the speed increment depending on the fuel supply and it is equal to 31.0174 [m/s]. The right picture shows the increment of an angle of attack depending on the fuel supply and it is equal to -0.1247 [rad]. The graphical presentation of a speed increment depending on the elevator is identical to the presentation in the Fig. 3, top right. A steady state of the speed increment is 15.6134 [m/s]. The graphical presentation of the increment of the angle of attack depending

on the elevator is identical to the presentation in the Fig. 3, bottom right. A steady state of the increment the angle of attack is -0.0721 [rad].

7.3 The hybrid distributed shared memory architecture

The hybrid distributed-shared memory system combines the advantages of architectures mentioned above. Each node consists of two processor cores.

Two nodes compute four independent mathematical models according to Eq. 17, 18, 23 and 24. The third node collects computed data from other nodes and shows the simulation results in a graphical form. Each computing node consists of Athlon X2 processor with two cores that share one memory. The core frequency is 2.6 GHz and the memory size is 2 GB. All nodes are interconnected via 1 Gbit/s Ethernet.

Graphical results represent their compatibility with previous forms of simulation and are shown in Fig. 5. The upper left picture shows the increment of speed depending on the fuel supply: it is equal to 31.0178 [m/s]. The upper right picture shows the increment of speed depending on the elevator: it is equal to -15.6151

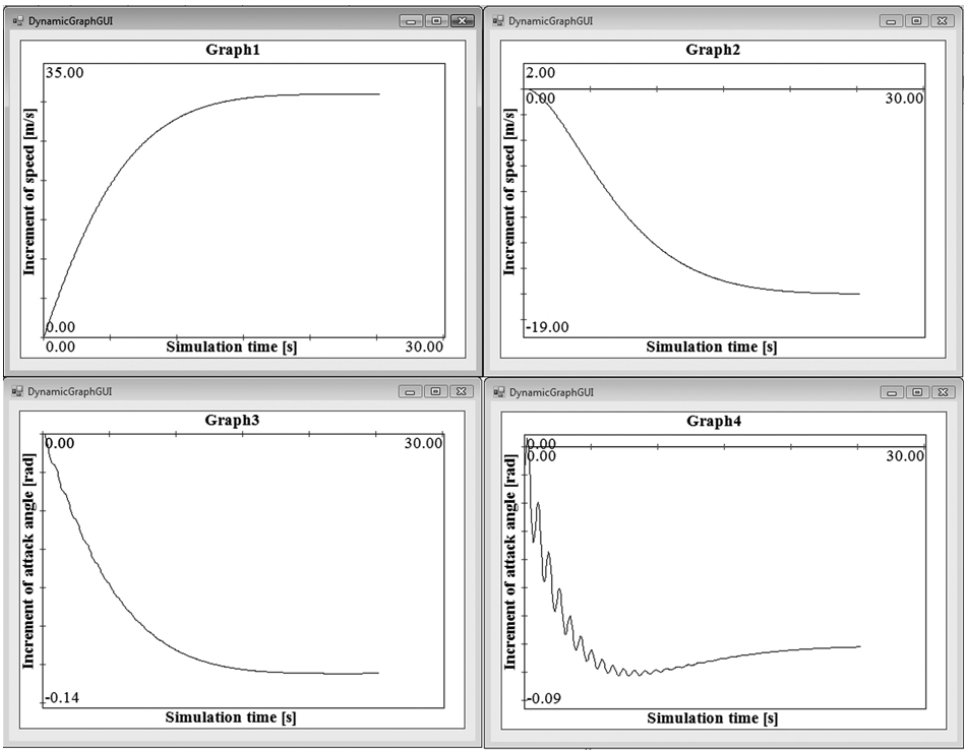

Figure 5: Screenshot of simulation results based on Eq. 17 – upper left, 18 – upper right, 23 – bottom left, 24 – bottom right.

[m/s]. The bottom left picture shows the increment of the angle of attack depending on the fuel supply: it is equal to −0.1248 [rad]. The bottom right picture shows the increment of the angle of attack depending on the elevator: it is equal to −0.0705 [rad].

8 Summary

The paper introduces three main architectures for efficient simulation of mathematical models: the DM, the SM and the hybrid one. The simulation is implemented on known different architectures that support parallel computing. This helps to overcome physical and architectural limitations of computational power that can be achieved with a single-processor system.

The use of a processor, a faster cache memory, operating memory access and a higher transmission capacity are then very suitable for the application. The hybrid architecture provides a higher transmission capacity and higher speed of computation.

Modelling of a parallel aspect of decomposed subsystems of a flight simulator in a form of a mathematical notation was carried out in accordance with Eq. 17, 18, 23 and 24. The results achieved in computation of mathematical models and the use of these three methods seem to be effective and pragmatic according to the results of from an integration algorithm.

The general accuracy from the comparison of simulation results of the use of the three methods is shown in Tab. 1, resulting in the accuracy less than 0.002.

Table 1: Accuracy of simulation results.

Type/ Model	**DM**	**SM**	**HYBRID**	**Accuracy** DM-SM	**Accuracy** DM-HYB	**Accuracy** SM-HYB
17	31.0192	31.0174	31.0178	0.0018	0.0014	−0.0004
18	−15.6142	−15.6134	−15.6151	0.0008	−0.0009	−0.0017
23	−0.1237	−0.1247	−0.1248	−0.0010	−0.0011	−0.0001
24	−0.0714	−0.0721	−0.0705	−0.0007	0.0009	0.0016

The strengths of the three models are also a combination of both the advantages: efficiency (memory savings) and ease of programming of a shared-memory method and scalability of a distributed-memory method. Sometimes we might get an advantage of faster simulation, but often on account of a defined quality. Generally, it is also possible to run a program faster because of such factors as availability of a better bandwidth for intra-node communication that provides extra communication

with the MPI across nodes. These facts and accuracy are very good for real-time simulation.

Acknowledgement: This work was supported by the Slovak Grant Agency for Science, by the VEGA 4/0330/09 grant.

References

Adhianto, L.; Banerjee, S.; Fagan, M.; Krentel, M.; Marin, G.; Mellor-Crummey, J., and Tallent, N. R. (2010): HPCTOOLKIT: tools for performance analysis of optimized parallel programs. *Concurrency Computat.: Pract. Exper.*, vol. 22, pp. 685–701. doi: 10.1002/cpe.1553.

Blakelock, J. H. (1991): *Automatic control of aircraft and Missiles*. Second Edition, John Wiley & Sons. Inc., New York.

Bajborodin, J. V. (1975): *Bortovyje sistemy upravlenja poletom*. Transport, Moskva.

Cellier, F., E.; Kofman, E. (2006): *Continuous System Simulation*. Basic Principles of Numerical Integation: New York : Springer, vol. 2, pp. 25–32.

Clark, R. N. (1996): *Control System Dynamics*. First Ed., Cambirdge University Press, New York, USA.

Duncan, S. H.; Gordon, P. L.; Zaluska, E. J.; Edwards, S. I. (1994): *Parallel processing in high integrity aircraft engine control*. Springer-Verlag, Berlin.

El-Rewini, H.; Abd-El-Barr, M. (2005): *Advanced Computer Architecture and Parallel Processing*. John Wiley & Sons, Inc., New York.

Hrubý, M.; Kočí, R.; Peringer, P.; Rábová, Z. (2002): Tools for creating of multimodels. *Kybernetes: The International Journal of Systems & Cybernetics*, vol. 9, pp. 1391–1400.

Huges, C.; Huges, T. (2003): *Parallel and Distributed Programming Using* C++. The Safari Press, Addison-Wesley Professional.

Chapman, B.; Jost, G.; Van der Pas, R. (2007): *Using OpenMP – Portable Shared Memory Parallel Programming*. The MIT Press, Massachusetts.

Chevance, R. J. (2005): *Server Architectures: Symetrical Multiprocessors*. Elsevier, Digital Press.

Krasovskij, A. A. (1950): *Sistemy avtomaticeskogo upravlenja poletom i ich analiticeskoje konstruirovanie*, Nauka, Moskva.

Kvasnica, P.; Páleník, T.; Čižmár, M. (2007): Mathematical model of aircraft and its visualization using MPI. *Proceedings of 3rd International Workshop on Grid Computing for Complex Problems, GCCP Bratislava*, pp. 117–125.

Lazar, T.; Adamčík, F.; Labún, J. (2007): *Modeling characteristics of the aircraft control*. First Ed. Technical University of Košice.

Martincová, P.; Grondžák, K.; Zábovský, M. (2008): *Programming in kernel of operating system Linux*. First Ed. University of Žilina.

McCormic, B. W. (1995): *Aerodynamics, Aeronautics and Flight Mechanics*. John Wiley & Sons, Inc., New York, Second Ed., USA.

Mmpich2 (2009): MPICH2 is a high performance and widely portable implementation of the Message Passing Interface (MPI) standard. [Online]. Available: http://www.mcs.anl.gov/mpi/mpich2.

Raeth P. G. (2010): Parallel MATLAB using standard MPI implementations. *Proceedings of High Performance Computing Modernization Program Users Group Conference (HPCMP–UGC), 2010 DoD*, pp. 438–441.

Rolfe J. M.; Staples K. J. (1986): *Flight Simulation*. Cambridge University Press, Cambridge.

Tereshenko V. (2009): One tool for building visual models. *Proceedings of Computational Intelligence, Modelling and Simulation, 2009. CSSim '09. International Conference on Brno*, pp. 59–62.