

NANO-MODELING OF RIBOSOMAL TRANSLATION PROCESS IMPLEMENTED IN A SUPERCOMPUTER PARALLEL PROCESSES ENVIRONMENT

Lech Znamirowski¹ and Olgierd A. Palusinski²

Abstract *¾ The paper presents a selected features of formal model of dynamics of the translation processes performed in a ribosome moving along the mRNA chain in a eukaryote cells. The model has a hierarchical, encapsulated architecture based on a message exchange fulfilled with the tRNA, rRNA and set of signals associated with the nucleus membrane. There is assumed an invariable structure of the DNA chain. A model using the concept of process communicating through messages paradigm is discussed. The computer network structure contains a workstation fulfilling the needs of Graphic User Interface, and a supercomputer working under control of the workstation to fulfill the tasks of time-consuming, interdependent numerical computations performed in the Message-Passing Interface environment on a UNIX platform. The hierarchical model has a form of shell containing the objects representing material elements (molecules or groups of molecules), and the objects representing messages, timely interdependent. The objects of the model have a form of the elements of specialized library.*

Index Terms *¾ Message-Passing Interface, Modeling, Parallel Computations, Ribosome.*

INTRODUCTION

The mechanism of protein synthesis is a process called translation because the string of letters of the four-letter alphabet of nucleic acids accordingly with genetic code is translated into string of amino acids of the 20-amino acid alphabet, forming proteins. The process takes place in a ribosome structure in presence of at least one kind of tRNA and activating enzyme for each amino acid. The translation processes is performed in a ribosome moving along the mRNA chain, and the activated precursor is driven by ATP. Protein synthesis takes place in initiation, elongation, and termination stages. The initiation stage results in the connecting of the initiator tRNA to the start signal in mRNA. The termination stage takes place when a stop signal in the mRNA is read by the protein release factor. Each nucleotide triplet, or codon, in mRNA chain encodes a specific amino acid. In the elongation stage, each molecule of tRNA binds only the amino acid proper to a particular codon, and tRNA recognize a codon by means of a

complementary nucleotide sequence named anticodon. The movement of the ribosome to the next codon is powered by the hydrolysis of GTP. When the termination stage occurs, the completed poly-peptid chain is released from the ribosome [1, 2].

The goal of model construction is investigation of dynamics of process translation parallelly, when the massive numerical procedures have to be performed [3]. The hierarchical model has a form of shell containing the objects representing material elements (molecules or groups of molecules), and the objects representing messages, timely interdependent. The objects of the model have a form of the elements of specialized library. In the paper, a network structure of the model and its timing features are described.

NETWORK STRUCTURE OF THE MODEL

The integrated network model performs two basic tasks in the two-computer system: the Graphic User Interface (GUI) is realized in the PC workstation, and the numerical computing needed in an application are performed in the multiprocessor based parallel computer. Time-consuming computations can be effectively shortened and the effective experiments performed by parallel realization of the numerical procedures. The computation process is controlled from the workstation working under Win98/2000 operating system, and a numerical computations are performed by the Sun Enterprise 6500 server (12 high performance, 64-bit SPARC™ [Scalable Processor ARChitecture] V9 RISC microprocessors with full throughput equal to 6 GFLOPS) working under the Solaris™ 7 Operating Environment [4].

MODEL TUNING

The proper understanding of the time dependencies between signals and material elements in the constructed model, needs experiments with the logic structure of model. Proper tool for these tasks is Hardware Description Language (HDL) with simulator. We will present two important features: communication between disk files and simulated variables, and timing generation for logic experiments.

Disk Communication

For extended logic model, good solution is language VHDL associated with environment Active-HDL [5, 6].

¹ Lech Znamirowski, Silesian University of Technology, Institute of Informatics, ul. Akademicka 16, 44-100 Gliwice, Poland
lznamiro@top.iif.polsl.gliwice.pl

² Olgierd A. Palusinski, University of Arizona, Department of Electrical and Computer Engineering, Tucson, AZ 85721 palusinski@ece.arizona.edu

In Active-HDL environment, the communication between disk files and a project variable can be performed basing on a listing presented below [7]. Module VHDL reads from disk four-bit vector named danewe.dat and next write it to the VHDL-variable named wektor. The same is for communication in opposite direction. The execution of described procedure depends only on the contents of procedure body .

```
--
-- File: c:\my designs\file_export\SRC\eximport.VHD
--{entity {eximport} architecture {copyinout} }
--
use std.textio.all;
--
entity eximport is
end eximport;
--
architecture copyinout of eximport is
begin
    process
        file wejście: text is in "c:\users\danewe.dat";
        file wyjście: text is out "c:\users\danewy.dat";
        variable linia1, linia2: line;
        variable wektor: bit_vector (3 downto 0);
        begin
            while not (endfile (wejście)) loop
                readline (wejście, linia1);
                read (linia1, wektor);
                write (linia2, wektor);
                writeline (wyjście, linia2);
            end loop;
        end process;
    end copyinout;
end architecture;
```

New possibility promise the AHDL (Analog Hardware Description Language) also informally known as VHDL-AMS (VHDL 1076.1 for "Analog and Mixed-Signal") [9, 10, 11], which can be directly used in a simulated mixed-mode systems.

Timing Synthesis

The timing for model control is mapped to the selected set of HDL statements [8]. Basic statements of VHDL used in mapping are a **process** (syntax according with [5]) statement:

```
process_statement ::=
    [process_label:]
    process (sensitivity_list)[is]
        process_declarative_part
    begin
        process_statement_part
    end process [process_label];
```

and a **if ... then** statement:

```
if_statement ::=
    [if_label:]
    if condition then
        sequence_of_statements
    {elseif condition then
        sequence_of_statements }
    [else
        sequence_of_statements]
    end if [if_label];
```

In a target code, the specified external signals in timing (inputs, outputs) are gathered as a ports of **entity** declaration, however, the declared internal signals of timing are placed into the declarative part of **architecture** body.

In Fig. 1, the activation of attribute of signal FPAA_I is presented (we assume e.g. DP1 signal for $t = t_{\text{moment of event}}$ is undetermined), and an effect of this event in a statement **if ... then** is following:

```
if (FPAA_I'event and FPAA_I='0') then
    DP1<='1';
```

so we have:

```
event   FPAA_I'event
result  FPAA_I='0'
then    DP1<='1'.
```

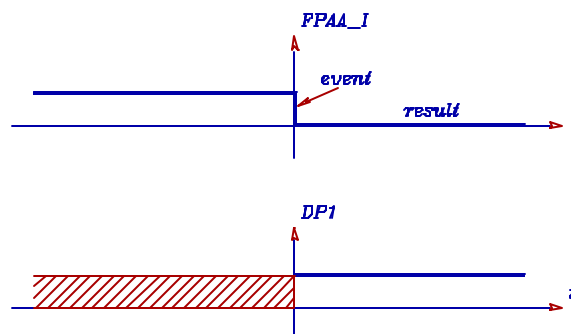


FIGURE. 1
THE ATTRIBUTE 'EVENT FOR AN EXEMPLARY SIGNAL FPAA_I.

An example of expression of the timing into the VHDL code, is a **process** P_A0 from listing below:

```
P_A0: process(RESET,R1_1,R2_2)
begin
    if (RESET'event and RESET='0') or (R2_2'event and
    R2_2='0')
    then A0<='0';
    elsif (R1_1'event and R1_1='0')
```

```

then A0<='1';
end if;
end process P_A0;

```

MESSAGE-PASSING INTERFACE PARADIGM IN THE MODEL CONSTRUCTION

The integrated network model integrates six main software modules distributed between a supercomputer (modules Control_Comp, Server_Comp, Programs_CC, and Programs_f77 characterized later) and a workstation working as a computation control unit (modules: Control_PC and Programs_C of the integrated network model).

Software tools

Completion of the system required adoption of existing stand alone numerical programs and writing a few new modules for system integration. Depending on proper platform [12, 13], it was necessary to use different compilers, scripts, protocols etc. In a system we have six main modules:

Control_PC - Control program in the integrated network model maintaining the main menu in the workstation, organizing operation on the input and output data in the computing process, and responsible for communication between operating systems in the two-computer integrated network model. Control_PC module resides on PC workstation – compiled in Visual C++ environment.

Control_Comp - Program working on a Unix platform. Control_Comp organizes computing process by division the data set into subsets, and ordering to the subsets separate processes. This function is fulfilled in the MPI (Message-Passing Interface) environment [14, 15]. Control_Comp module resides on Sun supercomputer – compiled with mpicc.

Compilation: /usr/local/mpi/bin/mpicc -o <name of the executable program> <name of the source program>.

Execution: /usr/local/mpi/bin/mpirun -np <number of processes> <name of the executable program>.

Server_Comp - Program for listening the demands of client in a client-server communication model. In the Server_Comp program are used the system function of the sockets interface [16]. Before starting any input/output operation, the system function socket() have to be used. System function bind() determined the name for non-name socket. After call a function socket(), client process adds a socket descriptor calling the system function connect() to set a connection with server. Function listen() declares readiness for connection receiving, next connection server can complete the system function accept(), to fulfill a request of the waiting client process. Server_Comp module resides on Sun supercomputer – compiled with CC compiler.

Programs_CC - Batch programs for numerical computing. Programs_CC modules reside on Sun supercomputer – compiled with CC compiler. This is done while working under control of Control_Comp.

Programs_f77 - Batch programs for numerical computing. Programs_f77 modules resides on Sun supercomputer – compiled with f77 compiler. Working under control of Control_Comp module.

Programs_C - Programs for numerical computing. Programs_C modules reside on PC workstation – compiled in Visual C++ environment.

Client-Server Communication Model

In the computer network applications, the standard model of data exchange is a client-server communication model. Server is a process waiting for requirement to connect from process called client to fulfill the determined tasks.

In integrated network model the selected communication protocol is TCP/IP. Firstly, layer TCP (Transmission Control Protocol) is responsible for correct delivery of data between client and server. In a case when data is lost, TCP initializes the retransmission until the data is completely and correctly received. This improve reliability of integrated network model (in a case of time-consuming computation lost of data is expensive). Secondly, login to the supercomputer can be performed from arbitrary computer working in the Internet. In integrated network model, the client workstation and computational server working in the network, communicate with each other using the package of subroutines that provide access to TCP/IP i.e. sockets interface [16].

Parallel Computations

To speed up time consuming computations, in the integrated network model, a supercomputer with 12 processors was used. The computations in multiprocessor's system are comparable to computations performed with parallel computations on multicomputer's system. In both cases the programs have to communicate with each other, and one process (or respectively program) have to supervise all operations. The difference is only in the type of communication. In multicomputer system, messages are sent through the computer network. In a case of multiprocessor system, the queues of messages resulting from processes' activity are created and supervisor process gather the waiting results together. The operating system of multiprocessor computer have to fulfill these requirements. The Solaris 7 Operating Environment is very well adapted to the tasks of this kind.

For the parallel computations, the numerical task of computations performed in the integrated network model, have to be moved from sequential to the parallel execution. In our case, the massive computation are divided between

eleven processors and indicate last processor (twelve) as a master gathering results. Each parallel process is ordered to the physical processor and the processes are mutually independent. Also, all variables in the processes are unique in each module. Data are gathered on a disk, and can be simultaneously read by processes. The parallel process of computing in the integrated network model is coded using MPI Environment by a program module Control_Comp. Message-Passing Interface is a specification for a library of routines to be called from ANSI C and Fortran 77 programs. Sending and receiving of messages between the parallel processes is basic operation in MPI environment for fulfilling the communication tasks. The basic point-to-point communication operations are *send* and *receive* implemented by the MPI functions MPI_Send() and MPI_Recv(), with proper (slightly different for the languages Fortran 77 and ANSI C, but functionally analogous) sets of arguments. In the integrated network model, the C code of the modul Control_Comp is divided into three parts, a common part for master and slave processes preparing data gotten from GUI (PC workstation), a second part for 11 slave processes sending the results (MPI_Send() function) of (the main part of time-consuming computation), and third part for master gathering results from 11 slave processes (MPI_Recv() function). The second and third part are surrounded with MPI environment activation functions MPI_Init() and MPI_Finalize(), the functions MPI_Comm_rank() and MPI_Comm_size() are used for processes initialization and find out the number of processes.

The internal inter-process' communication in Control_Comp module is based on MPI_DOUBLE C datatype and MPI_COMM_WORLD predefined communicator [14, 17].

CONCLUSIONS

The paper presents a selected features of formal model used for modeling composite processes. The problems of timing tuning of model and the distribution of two-computer resources is discussed. Presented integrated network model is framework for computer simulations fully using network computer system's features, where depending on task division between resources of a system, it is possible to build friendly user interfaces, and independently improve system performance. In implemented system, through modules integration and parallel operation, the time of the system interaction was considerably shortened. Integrity of the system is satisfied through the compatibility of the data structures, control of the system from main graphic application, and automatic transfer of data between PC workstation and the supercomputer through the network. The user can get results which are basic for design, but it also is possible to access to intermediate results of computation. This is necessary for checking the correctness of actual algorithms as well as in future development of the system's capabilities.

REFERENCES

- [1] Green R., H. F. Noler, "Ribosomes and Translation", *Annu. Rev. Biochem.*, Vol. 66, 1997, pp. 679-716.
- [2] Stryer L., *Biochemistry*, W, H. Freeman and Company, New York, 1994.
- [3] Clote P., R. Backofen, *Computational Molecular Biology*, John Wiley & Sons, Ltd, Chichester 2000.
- [4] Sun Enterprise 6500, "On-line", System Documentation, 157.158.1.1:8888 (address:port).
- [5] "IEEE Standard VHDL Language Reference Manual", IEEE Std 1076-1993, IEEE, New York, June 1994.
- [6] "Active-HDL, ver. 3.5", ALDEC, Henderson 1998.
- [7] Znamirowski L., "CAD/CAE for Hardware Encryption of Files", *STUDIA INFORMATICA*, SUT Press, Vol. 21, No. 1 (39), 2000, pp. 549-566, (in Polish).
- [8] Znamirowski L. and O. A. Palusinski, "HDL Code Generation for Implementation of Timing Control in Mixed-Mode Systems", *2000 Western MultiConference, (Intern. Conf. on Simulation and Multimedia in Engr. and Edu.)*, The Society for Computer Simulation International, Vol. 32, No. 1. San Diego, January 23-27 2000.
- [9] Vachoux A., "Analog and Mixed-Signal Extension to VHDL", *Analog Integrated Circuits and Signal Processing*, Kluwer Academic Publishers, No. 16, 1998, pp. 185-200.
- [10] "VHDL-AMS Language Architecture, VHDL-AMS Proposal", host: ftp.uu.net, path: /doc/standards/ ieeec/1076.1/analog, 1996.
- [11] "VHDL 1076.1: Analog and Mixed Signal Extension for VHDL", IEEE Design Automation Standard Committee 1076.1 WG.
- [12] Microsoft Corp., "MSDN, the Microsoft Developer Network", 3 x CD-ROM , Technical Documentation for Microsoft Visual Studio 6.0, Redmond, October 1998.
- [13] Stevens W. R., "Programming of Network Application in a Unix System", *WNT*, Warszawa 1995 (in Polish).
- [14] Gropp W., E. Lusk, and A. Skjellum, "Using MPI", *The MIT Press*, Cambridge, Massachusetts, London, England 1994.
- [15] "Using MPI in parallel programming", <http://www.csc.fi/english/>.
- [16] Foster I., "Designing and Building Parallel Programs", 1995, http://prioris.im.pw.edu.pl/~mstefanczyk/pdp_book/node6.html.
- [17] Seweryn K., O. A. Palusinski, L. Znamirowski, "Network Super-computer Implementation in Massive CAD for VLSI Computations ", *STUDIA INFORMATICA*, Vol. 22, No. 5 (43), 2001 (in printing).