33rd Annual International Conference of the IEEE EMBS
Boston, Massachusetts USA, August 30 - September 3, 2011

A General CellML Simulation Code Generator using ODE Solving Scheme Description

Akira AMANO, Naoki SOEJIMA, Takao SHIMAYOSHI, Hiroaki KUWABARA, Yoshitoshi KUNIEDA

***Abstract*— To cope with the complexity of the biological function simulation models, model representation with description language is becoming popular. However, simulation software itself becomes complex in these environment, thus, it is difficult to modify target computation resources or numerical calculation methods or simulation conditions. Typical biological function simulation software consists of 1) model equation, 2) boundary conditions and 3) ODE solving scheme. Introducing the description model file such as CellML is useful for generalizing the first point and partly second point, however, third point is difficult to handle. We introduce a simulation software generation system which use markup language based description of ODE solving scheme together with cell model description file. By using this software, we can easily generate biological simulation program code with different ODE solving schemes. To show the efficiency of our system, experimental results of several simulation models with different ODE scheme and different computation resources are shown.**

I. Introduction

Experiments of many physical phenomena are now conducted using computer simulation. Biological functions or system are also being examined by simulations, however, biological function simulations are different from others in the sense of model complexity. This feature results in complex simulation program which are very difficult to maintain, modify and expand. One method of coping with the model complexity is to use some markup language based model descriptions. One of the most famous modeling language is CellML [2] which is intended to be used for describing cellular function models. Many models are already provided by the CellML repository [1]. To use CellML models, simulation softwares which can handle these files are necessary [4][5].

A typical simulation program of cellular or biological function model consists of three parts: one is model equation itself, second is boundary conditions, and third is ODE solving scheme such as Euler method or Runge-Kutta method. In some case, researchers want to evaluate more sophisticated ODE solving scheme for speedup or stabilizing the numerical calculation. In other case, researchers want to use parallel computation resources for large scale simulations. In another case, coupling simulation between different phenomena such as structural mechanics or fluid dynamics together with cellular electrophysiology are necessary. In all of these cases, special knowledge and programming skills are necessary to construct or modify simulation programs even if the original software can handle CellML files.

A. Amano is with Department of Bioinformatics, Ritsumeikan Univerisity, Shiga-ken, 525-8577, Japan (phone:+81-77-561-2584; e-mail:a-amano@fc.ritsumei.ac.jp)

N. SOEJIMA, H. KUWABARA, Y. KUNIEDA is with Department of Informatics, Ritsumeikan Univerisity, Shiga-ken, 525-8577, Japan)

T. Shimayoshi is with ASTEM RI, Kyoto, 600-8813, Japan

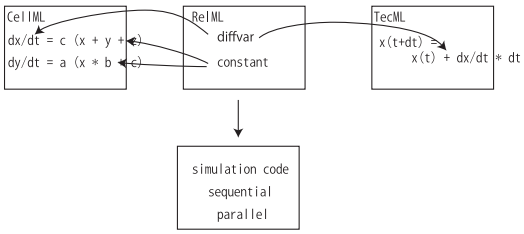

Fig. 1. Inputs and outputs of proposed system.

To cope with this problem, we designed a system which can automatically generate complex simulation program for special computation resources by using cell model written by CellML, and ODE solving scheme written by a description language.

II. General Purpose Simulation Code Generator

A. Simulation Code Generation

In our system, cell model is provided by a CellML[2] file. We designed a description language called TecML (Time Evolution Calculation Markup Language) to describe ODE solving schemes, and a TecML file is also provided to the system. Since the type of each variable in cell model changes according to the experimental protocol, the variable type information is also necessary for the simulation. This information should be written in a experimental protocol description, however, the design of the experimental protocol description is still under research in our system. Therefore, we designed a simple description language RelML (Relation Markup Language) which describes the relation between a CellML file and a TecML file(Fig.1).

From these information, our system generates simulation program for single CPU environment and CPU with parallel processing device (GPGPU). Note that, some cell model requires reordering of equations written in CellML file. Also in some cell model, solving of linear or nonlinear simultaneous equations are required. This happens because a CellML file is only a static description of a cell model and separated from the numerical calculation method. One of the goal of our system is to automatically cope with this problem, however, this aspect is still under research, thus very simple equation reordering is only provided.

978-1-4244-4122-8/11/$26.00 ©2011 IEEE

```
<tecml>
  <inputvar name="xi" type="diffvar" />
  <outputvar name="xo" type="diffvar" />
  <variable name="d" type="deltatimevar" />
  <variable name="t" type="timevar" />
  <variable name="x1" type="diffvar" />
  <variable name="y1" type="arithvar" />
  <variable name="k1" type="derivativevar" />
  <variable name="z" type="constvar" />
  <function name="f" type="diffequ">
    <argument type="diffvar" />
    <argument type="timevar" />
    <argument type="arithvar" />
    <argument type="constvar" />
  </function>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
      <eq/>
      <ci>x1</ci>
      <ci>xi</ci>
    </apply>
  </math>
  ...
</tecml>
```

Fig. 2. A TecML file example (part).

```
<relml>
  <cellml filename="model/cellml/sample.cellml">
  <tecml filename="model/tecml/Euler.tecml">
  <variable component="environment"
            name="time" type="timevar" />
  <variable component="membrane"
            name="V" type="diffvar" />
  <variable component="membrane"
            name="I_stim" type="arithvar" />
  <variable component="membrane"
            name="R" type="constvar" />
</relml>
```

Fig. 3. A RelML file example (part).

B. Description Languages

1) CellML: Cell Model Description Language: CellML [2] is widely used as a description language of cell models. We used a CellML file as a cell model input file.

2) TecML: Time Evolution Calculation Markup Language: TecML is a description language which is used to describe ODE solving scheme such as Euler method, Modified Euler method or 4th explicit Runge-Kutta method. In a TecML file, variables used in ODE solving scheme equations are declared with variable type. The variable type is one of time, delta time, differential variable, arithmetic variable, derivative variable or constant. Also the input and the output differential variables are declared. The name of function which represents function to calculate derivatives and arithmetic variables are also declared. Finally, equations to calculate output variable from input variable are written with mathml. An example TecML file (part) is shown in Fig.2.

3) RelML: Relation Markup Language: RelML is a description language which is used to describe a relation between a CellML file and a TecML file. Thus one RelML file is necessary for each combination of a CellML and a TecML files. In a RelML file, first filenames of both CellML and TecML are described. All the variable names and their types are declared. The variable type is one of differential variable, arithmetic variable, constant and time variable which corresponds to that of TecML. An example of RelML file is shown in Fig.3.

TABLE I

INFORMATION WRITTEN IN CELL MODEL DESCRIPTION LANGUAGES

notation	meanings
$\boldsymbol{x} = [x_1, x_2, \ldots, x_{N_x}]^T$	differential variables
$\boldsymbol{k} = [k_1, k_2, \ldots, k_{N_x}]^T$	derivative variables
$\boldsymbol{y} = [y_1, y_2, \ldots, y_{N_y}]^T$	arithmetic variables
$\boldsymbol{z} = [z_1, z_2, \ldots, z_{N_z}]^T$	constants
t	time
$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}) = [f_1(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}), f_2(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}), \ldots, f_{N_x}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})]^T$	equations with $\boldsymbol{k}$ in LHS
$\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}) = [g_1(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}), g_2(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}), \ldots, g_{N_y}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})]^T$	equations with $\boldsymbol{y}$ in LHS

III. CODE GENERATION ALGORITHM

A. Cell Model Description

Information written in a cell model file can be summarized as shown in Table I.

Let a cell model be written as follows,

$$\begin{cases} \boldsymbol{k} = \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}) \\ \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}) \end{cases} \tag{1}$$

where, $\langle^{\bullet}\rangle$ denotes differentials by t (d/dt). Here, we denote $\boldsymbol{x}$ as differential variable vector, $\boldsymbol{y}$ as arithmetic variable vector. $\boldsymbol{z}$ is constant vector and t denotes time. Variables $\boldsymbol{x}$,$\boldsymbol{y}$,$\boldsymbol{z}$ can be denoted as follow.

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_{N_x}]^T \tag{2}$$

$$\boldsymbol{y} = [y_1, y_2, \ldots, y_{N_y}]^T \tag{3}$$

$$\boldsymbol{z} = [z_1, z_2, \ldots, z_{N_z}]^T \tag{4}$$

$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})$ is a function vector whose LHS's are $\dot{\boldsymbol{x}}$ where $\dot{x}_i = f_i(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})$.

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}) = [f_1(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}), f_2(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}), \ldots, f_{N_x}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})]^T \tag{5}$$

$\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})$ is a function vector whose LHS's are $\boldsymbol{y}$ where $y_i = g_i(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})$.

$$\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}) = [g_1(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}), g_2(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z}), \ldots, g_{N_y}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})]^T \tag{6}$$

B. ODE solving scheme description

Information written in a TecML file can be summarized as shown in Table II.

Let differential equations of a cell model be denoted by $d\boldsymbol{\xi}/dt = \boldsymbol{\Phi}(\boldsymbol{\xi}, \boldsymbol{\iota}, t), \boldsymbol{\iota} = \boldsymbol{\Gamma}(\boldsymbol{\xi}, \boldsymbol{\iota}, t)$. The relation between the differential variable value $\boldsymbol{\xi}_0$ (time t) and $\boldsymbol{\xi}_{N_\xi}$ (time $t + \delta$ in a TecML file can be denoted as follow.

$$\boldsymbol{\xi}_i = \boldsymbol{\sigma}_i(\Xi, K, \delta)\ (1 \le i \le N_\xi) \tag{7}$$

$$\boldsymbol{\kappa}_i = \boldsymbol{\Phi}(\boldsymbol{\xi}_{i-1}, \boldsymbol{\iota}_{i-1}, \tau_{i-1})\ (1 \le i \le N_\xi) \tag{8}$$

$$\boldsymbol{\iota}_i = \boldsymbol{\Gamma}(\boldsymbol{\xi}_i, \boldsymbol{\iota}, \tau_i)\ (0 \le i \le N_\xi) \tag{9}$$

$$\tau_i = T_i(t, \delta)\ (0 \le i \le N_\xi)$$

where,

$$\Xi = [\boldsymbol{\xi}_0\ \boldsymbol{\xi}_1\ \ldots\ \boldsymbol{\xi}_{N_\xi}]^T \tag{10}$$

$$K = [\boldsymbol{\kappa}_1\ \boldsymbol{\kappa}_2\ \ldots\ \boldsymbol{\kappa}_{N_\xi}]^T. \tag{11}$$

TABLE II

INFORMATION WRITTEN IN TIME EVOLUTION CALCULATION ML

notation	meanings
$\boldsymbol{\xi}_0$	current differential variable vector
$\boldsymbol{\xi}_{N_\xi}$	output differential variable vector
t	variable for time
δ	variable for time step
ζ	constant
$\boldsymbol{\xi}_i$ $(1 \leq i \leq N_\xi)$	temporal differential variable vector
$\boldsymbol{\kappa}_i = \Phi(\boldsymbol{\xi}_{i-1}, \boldsymbol{\iota}_{i-1}, \tau_{i-1})$ $(1 \leq i \leq N_\xi)$	vector variable for derivative variables
$\boldsymbol{\iota}_i = \boldsymbol{\Gamma}(\boldsymbol{\xi}_i, \boldsymbol{\iota}, \tau_i)$ $(0 \leq i \leq N_\xi)$	temporal variable from $\boldsymbol{\xi}$
$\tau_i = T_i(t, \delta)$ $(1 \leq i \leq N_\xi)$	temporal variables for time
$\Xi = [\boldsymbol{\xi}_0, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_{N_\xi}]^T$	vector for $\boldsymbol{\xi}$
$K = [\boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2, \ldots, \boldsymbol{\kappa}_{N_\xi}]^T$	vector for $\boldsymbol{\kappa}$
$\sigma_i(\Xi, K, \delta)$	relation between $\boldsymbol{\xi}_i$ and Ξ, K, δ

TABLE III

INFORMATION WRITTEN IN RELML FILE

relation	meanings
$\boldsymbol{\xi}_k = [\xi_{k,1}, \xi_{k,2}, \ldots, \xi_{k,N_\kappa}]^T$ $\Leftrightarrow \boldsymbol{x} = [x_1, x_2, \ldots, x_{N_x}]^T$	differential variable vector
$\boldsymbol{\kappa}_k = [\kappa_{k,1}, \kappa_{k,2}, \ldots, \kappa_{k,N_\kappa}]^T$ $\Leftrightarrow \boldsymbol{k} = [k_1, k_2, \ldots, k_{N_x}]^T$	differential derivative vector
$\boldsymbol{\iota}_k \Leftrightarrow \boldsymbol{y} = [y_1, y_2, \ldots, y_{N_y}]^T$	temporal variable vector
$d\boldsymbol{\xi}/dt = \Phi(\boldsymbol{\xi}, \boldsymbol{\iota}, t) \Leftrightarrow \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})$	differential equation
$\boldsymbol{\iota} = \Gamma(\boldsymbol{\xi}, \boldsymbol{\iota}, t) \Leftrightarrow \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})$	arithmetic equation

C. Description of relation between cell model information and tecml information

The relation between the information of a cell model and a tecml scheme are described in a RelML file which are summarized in Table III.

The elements of differential variables in TecML are identical to those of cell model, therefore, there exist corresponding variable $\boldsymbol{x}$ for each $\boldsymbol{\xi}$.

$$\boldsymbol{\xi}_k = \boldsymbol{x}_k = [x_{k,1}, x_{k,2}, \ldots, x_{k,N_x}]^T \tag{12}$$

The elements of derivative variables $\boldsymbol{\kappa}$ in a TecML file has corresponding derivative variables $\dot{\boldsymbol{x}}$ in a cell model.

$$\boldsymbol{\kappa}_k = \dot{\boldsymbol{x}}_{\boldsymbol{k}} = [\dot{x}_{k,1}, \dot{x}_{k,2}, \ldots, \dot{x}_{k,N_x}]^T \tag{13}$$

The elements of arithmetic variable $\boldsymbol{\iota}$ in a TecML file has corresponding arithmetic variables $\boldsymbol{y}$ in a cell model.

$$\boldsymbol{\iota}_k = \boldsymbol{y}_k = [y_{k,1}, y_{k,2}, \ldots, y_{k,N_y}]^T \tag{14}$$

The actual calculation equations $d\boldsymbol{\xi}/dt = \boldsymbol{\Phi}(\boldsymbol{\xi}, \boldsymbol{\iota}, t), \boldsymbol{\iota} = \Gamma(\boldsymbol{\xi}, \boldsymbol{\iota}, t)$ in a TecML file correspond to the model equations $\boldsymbol{f}, \boldsymbol{g}$ in cell model.

$$\phi_k(\boldsymbol{\xi}, \boldsymbol{\iota}, \tau) = f_k(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})|_{\boldsymbol{x}=\boldsymbol{\xi}, \boldsymbol{y}=\boldsymbol{\iota}, t=\tau} \tag{15}$$

$$\gamma_k(\boldsymbol{\xi}, \boldsymbol{\iota}, \tau) = g_k(\boldsymbol{x}, \boldsymbol{y}, t, \boldsymbol{z})|_{\boldsymbol{x}=\boldsymbol{\xi}, \boldsymbol{y}=\boldsymbol{\iota}, t=\tau} \tag{16}$$

D. Algorithm

By using the information written in a cell model file, a TecML file and a RelML file, simulation program can be generated by the algorithm shown in Fig.4.

Note that, the subroutine replace_v() replaces variables in the TecML equation with program variables, replace_sj() generates one scalar equation from vector equation, replace_d()

```
generate_equations() {
    string equation_set[] = all equations in TecML ;
    string equation_set2[] = null ;
    int equ2_count = 0 ;
    for (i ∈ 1 ...number of equations in equation_set)) {
        equation_set[i] = replace_v(equation_set[i]);
        if (equation_set[i] includes f ) {
            for (j ∈ (1 ... N_x ))
                equation_set2[equ2_count++] = replace_sj(equation_set[i], j);
        } else if (equation_set[i] includes g ) {
            for (j ∈ (1 ... N_y ))
                equation_set2[equ2_count++] = replace_sj(equation_set[i], j);
        } else {
            for (j ∈ (1 ... N_x ))
                equation_set2[equ2_count++] = replace_d(equation_set[i], j);
        }
    }
    for (i ∈ (1 ...number of equations in equation_set2)){
        if (equation_set2[i] includes f ) {
            for (j ∈ (1 ... N_x ))
                equation_set2[i] = replace_f(equation_set2[i], j);
        } else if (equation_set2[i] includes g ) {
            for (j ∈ (1 ... N_y ))
                equation_set2[i] = replace_g(equation_set2[i], j);
        }
    }
    output all equation_set2;
}
replace_v(string equ) {
    for (i ∈ (0..N_ξ) ) {
        replace all ξ_i with x_i in equ;
        replace all κ_i with k_i in equ;
        replace all ι_i with y_i in equ;
    }
    replace all τ with t in equ;
    replace all δ with d in equ;
    replace all ζ with z in equ;
    return equ;
}
replace_sj(string equ, j) {
    string lhs = left hand side of equ;
    string rhs = right hand side of equ;
    replace all k_i with k_{i,j} in lhs;
    replace all y_i with y_{i,j} in lhs;
    replace all Φ with φ_j in rhs;
    replace all Γ with γ_j in rhs;
    equ = lhs + "=" + rhs;
    return equ;
}
replace_d(string equ, j) {
    replace all x_i with x_{i,j} in equ;
    replace all k_i with k_{i,j} in equ;
    equ = lhs + "=" + rhs;
    return equ;
}
replace_f(string equ, j) {
    string lhs = left hand side of equ;
    string rhs = right hand side of equ;
    replace φ_j(x_k, y_l, t) with f_j(x, y, t, z) in rhs;
    for (i ∈ (0..N_x) )
        replace all x_i with x_{k,i} in rhs;
    for (i ∈ (0..N_y) )
        replace all y_i with y_{l,i} in rhs;
    equ = lhs + "=" + rhs;
    return equ;
}
replace_g(string equ, j) {
    string lhs = left hand side of equ;
    string rhs = right hand side of equ;
    replace γ_j(x_k, y_l, t) with g_j(x, y, t, z) in rhs;
    for (i ∈ (0..N_x) )
        replace all x_i with x_{k,i} in rhs;
    for (i ∈ (0..N_y) )
        replace all y_i with y_{l,i} in rhs;
    equ = lhs + "=" + rhs;
    return equ;
}
```

Fig. 4. Code Generation Algorithm

```
cell model
$r = ( $x * $x * $x )
( d$x / ( d$t ) ) = ( $c * ( ( $x - ( $r / 3.0 ) ) + $y + $z ) )
( d$y / ( d$t ) ) = ( ( ( $a - $x ) - ( $b * $y ) ) / $c )
```

Fig. 5. cell model example

```
variable definition
differential variables : { &xi , &x1, &xo }
derivative variables : { &k1 , &k2 }
arithmetic variables : { &y1 , &y2 }
constant : { &z }

equation definition
&y1 = g ( &xi , &t , &y1 , &z )
&k1 = f ( &xi , &t , &y1 , &z )
&x1 = &xi + ( &k1 * &d )
&y2 = g ( &x1 , &t , &y2 , &z )
&k2 = f ( &x1 , &t , &y2 , &z )
&xo = &xi + ( ( &d / 2 ) * ( &k1 + &k2 ) )
```

Fig. 6. TecML example (Euler)

generates multiple scalar equations from vector equation which do not contain function f or g, replace_f() unfolds function f and replace_g() unfolds function g.

By providing the cell model (Fig.5), ODE solving scheme with TecML (Fig.6) and relation between them with RelML (Fig.7), the algorithm could successfully generate ODE solving program as shown in Fig.8.

IV. Experimental Results

A. Conditions

We used Kyoto model [3] for the experiments. The model is a cardiac cell model of small mammalian, and can calculate action potential and ion concentrations by ODEs.

Calculation program of 1) single cell model, 2) multi cell model with no intercellular interaction and 3) multi cell model with excitation propagation calculation were generated by using a) Euler method, b) Modified Euler method and c) 4th order explicit Runge-Kutta method. Double precision variables were used for the differential variables. In each experiment, cell model was calculated for 400 [msec] with time step of 0.01 [msec] unless noted. Resulting membrane potential was recorded with interval 1.0 [msec]. Computational resources are summarized in Table IV.

```
cell model variable definition
differential variables : { $x , $y}
arithmetic variables : { $r}
constants : { $a , $b , $c , $z}
```

Fig. 7. RelML example

$$
\begin{aligned}
&y1[0] = (xi[0] * xi[0] * xi[0])\\
&k1[0] = (z[2] * ((xi[0] - (y1[0]/3.0)) + xi[1] + z[3]))\\
&k1[1] = (((z[0] - xi[0]) - (z[1] * xi[1]))/z[2])\\
&x1[0] = xi[0] + (k1[0] * d)\\
&x1[1] = xi[1] + (k1[1] * d)\\
&y2[0] = (x1[0] * x1[0] * x1[0])\\
&k2[0] = (z[2] * ((x1[0] - (y2[0]/3.0)) + x1[1] + z[3]))\\
&k2[1] = (((z[0] - x1[0]) - (z[1] * x1[1]))/z[2])\\
&xo[0] = (xi[0] + ((d/2) * (k1[0] + k2[0])))\\
&xo[1] = (xi[1] + ((d/2) * (k1[1] + k2[1])))
\end{aligned}
$$

Fig. 8. Output of the algorithm with information of Fig.5,Fig.6, Fig.7

TABLE IV
COMPUTATIONAL RESOURCES

Host	CPU	Core i7 930 2.80GHz
	Main Memory	6GB
Device	GPU	Tesla C2050
	VRAM	3GB

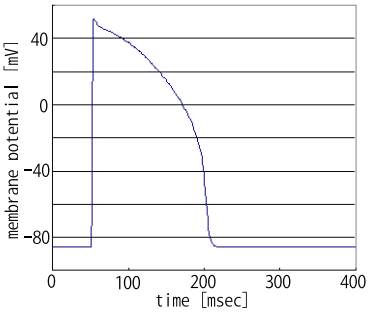

Fig. 9. Action Potential of Kyoto Model

B. Single Cell Model

The number of equations to calculate time evolution and the size of transfer data are summarized in Table V.

TABLE V
NUMBER OF EQUATIONS AND SIZE OF TRANSFER DATA

ODE scheme	number of equations	size of data transfer(Byte)
Euler	336	296
Runge-Kutta	1010	296

Resulting action potential is shown in Fig.9 which is identical to the figure in the original Kyoto model paper [3].

Next, numerical calculation error was evaluated for the ODE solving scheme of 1st order Euler method, 2nd order Modified Euler method and 4th order explicit Runge-Kutta method by using Kyoto model. Note that the numerical calculation results using high precision math library was used as the ground truth. The relative error were evaluated for time step 0.01 – 0.0001 [msec] which are shown in Fig.10.

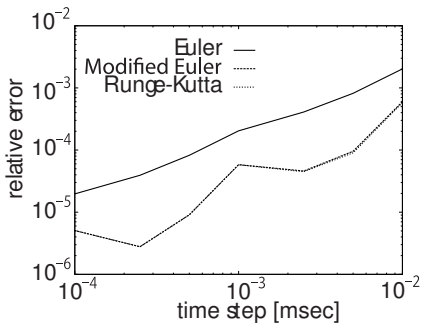

Fig. 10. Calculation Errors of Euler, Modified Euler and Runge Kutta methods for Kyoto Model

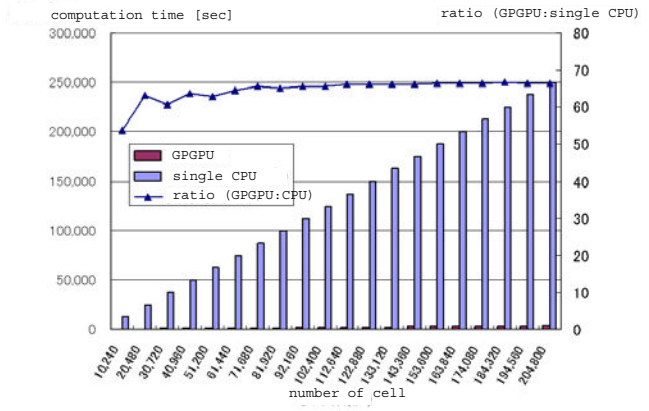

Fig. 11. Computation time for multi cell model with single CPU and GPGPU.

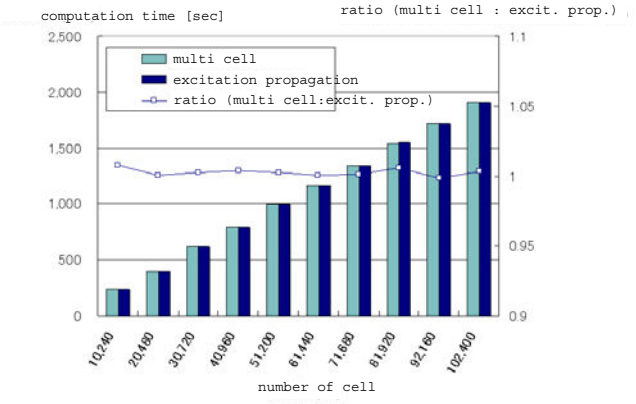

Fig. 12. Calculation time of multi cell model and multi cell excitation propagation model

C. Multi Cell Model

Multi cell model was used for evaluating the acceleration by GPGPU. In this experiment, 4th order explicit Runge-Kutta method was used for ODE solving scheme. The number of cells were 10240 to 204800. Resulting computation times are shown in Fig.11.

We can find that the ratio between the GPGPU computation time and the single CPU converge to a value which was about 66 in this case. This value changes according to the simulation code or computation resources. Note that about 60% of the computation time was used for data transfer between the GPGPU memory and CPU memory and less than 40% of the computation time was used for the ODE calculation.

D. Excitation Propagation Calculation Model

In the multi cell model, data reference between different cells do not happen, however, in the typical parallel calculation, data reference among large data occurs which results in low calculation speed. To evaluate this aspect, we modified the multi cell model to calculate excitation propagation of single cardiac cell fiber. Since the description and code generation of field related phenomena is still in progress in our project, we manually introduced the code to the multi cell model program code.

The resulting computation time with multi cell model computation times are shown in Fig.12. In this case, the difference between the computation time of both model was very small. This is because all the model variables were stored in the GPGPU memory in this case.

V. Conclusions

We proposed a biological function simulation program code generator using model description file together with ODE solving scheme description file. By using this system, we can easily generate simulation program with complex ODE solving scheme and also compare several ODE solving schemes. By preparing a parallel code generator for this system, we also can easily generate simulation code for parallel processing environments such as GPGPU.

The system will be publicly available in our project website.

References

[1] Cellml model repository. http://models.cellml.org/cellml.
[2] Cuellar AA, Lloyd CM, Nielsen PF, Bullivant DP, Nickerson DP, and Hunter PJ. An overview of cellml 1.1, a biological model description language. *SIMULATION*, 79(12):740–747, 2003.
[3] Satoshi Matsuoka, Nobuaki Sarai, Shinobu Kuratomi, Kyoichi Ono, and Akinori Noma. Role of individual ionic current systems in ventricular cells hypothesized by a model study. *JJP*, 53:105–123, 2003.
[4] David P. Nickerson, Alberto Corrias, and Martin L. Buist. Reference descriptions of cellular electrophysiology models. *Bioinformatics*, 24(8):1112–1114, 2008.
[5] Missan S and McDonald TF. Cese: Cell electrophysiology simulation environment. *Appl Bioinformatics*, 4(2):155–6, 2005.