IPSJ Transactions on System LSI Design Methodology Vol. 3 244–256 (Aug. 2010)

Regular Paper

Automatic Pipeline Construction Focused on Similarity of Rate Law Functions for an FPGA-based Biochemical Simulator

Hideki Yamada,[†1] Yui Ogawa,[†1] Tomonori Ooya,[†1]
Tomoya Ishimori,[†1] Yasunori Osana,[†2]
Masato Yoshimi,[†3] Yuri Nishikawa,[†4]
Akira Funahashi,[†4] Noriko Hiroi,[†4]
Hideharu Amano,[†4] Yuichiro Shibata[†1]
and Kiyoshi Oguri[†1]

For FPGA-based scientific simulation systems, hardware design technique that can reduce required amount of hardware resources is a key issue, since the size of simulation target is often limited by the size of the FPGA. Focusing on FPGA-based biochemical simulation, this paper proposes hardware design methodology which finds and combines common datapath for similar rate law functions appeared in simulation target models, so as to generate area-effective pipelined hardware modules. In addition, similarity-based clustering techniques of rate law functions are also presented in order to alleviate negative effects on performance for combined pipelines. Empirical evaluation with a practical biochemical model reveals that our method enables the simulation with 66% of the original hardware resources at a reasonable cost of 20% performance overhead.

1. Introduction

Systems biology, an attempt to analyze and understand the mechanism of life phenomena in a system level has now become more active reflecting recent advance in life science and rapid accumulation of quantitative data obtained by biological experiments. Also a demand for high performance biochemical simulators is increasing, since simulating practical kinetic models of biochemical pathways is a very time consuming process [1),2)].

A major difficulty in accelerating biochemical simulation comes from diversity and heterogeneousness of simulation models. There is no single governing equation in biochemical simulations, but every different model consists of a variety of different equations. Therefore, we had launched a project to implement FPGA-based biochemical simulator called ReCSiP (ReConfigurable Cell Simulation Platform) [3),4)], whose hardware structure can be tailored to fit each simulation model.

A simulation target of ReCSiP is given in a set of an ordinary differential equation (ODE)-based chemical reaction model described in systems biology markup language (SBML) [5)]. Then the equations for the model are automatically implemented as custom hardware on an FPGA and high-throughput simulation is carried out. A key issue for such FPGA-based systems is how desired functional units are compactly implemented, since the more functional units implemented on a chip, the larger the model we can simulate. Also, the benefit of parallel processing increases by replicating the implementation of small hardware modules.

The proposed method developed in this work focuses on this problem. To make the simulation within limited circuit resources on an FPGA possible, the method automatically finds the common datapath from rate law functions used in a given target biochemical model, and then generates compact pipelined hardware for simulation by combining and sharing the common datapath. In order to alleviate the performance degradation by sharing, methods for clustering the rate law functions into similar groups are also presented.

The work described in the paper is about hardware resource reduction schemes on ReCSiP. In our early work, manually designed hardware libraries for frequently used rate law functions called SBML predefined functions [6)] were implemented by merging the common datapath [7)]. Automatic datapath combining method for any arbitrary pair of rate law functions was presented in our previous work [8)], but the effectiveness of the method was not evaluated in an empirical way thus the impact on the whole simulation performance was not revealed yet.

On the other hand, a wide variety of hardware resource reduction techniques with the common datapath sharing have been proposed in the field of electronic

†1 Nagasaki University
†2 Seikei University
†3 Doshisha University
†4 Keio University

design automation and a lot of novel methods for reconfigurable systems have been developed in recent years. Reference 9) formulated datapath merging as integer programming under a few restrictions. Reference 10) presented a circuit area reduction method which extracts common subgraphs in a circuit level and combines them in a technology mapping stage of FPGA implementation. Reference 11) showed a fast heuristic method to merge multiple datapath considering similarity. The effect of this method was further improved by adding novel optimization techniques in a high-level synthesis process [12]. However, these existing methods are not suitable to be applied to deeply pipelined hardware modules. Since these methods do not consider dependencies between datapaths, combined hardware is able to work as only one of the shared functionality at one time, which will result in severe throughput degradation due to frequent pipeline stalls. Another issue is that the existing methods straightforwardly generate only one module that contains all given datapaths, without considering which datapaths should be combined. Putting too many functionality into one hardware module often leads performance degradation in an application level [7]. To cope with these problems, we present a datapath merging method which allows combined hardware to simultaneously operate multiple functionality in a pipelined manner, as well as a similarity-based datapath clustering method to determine which datapath to be combined. Also this paper presents empirical evaluation with a practical biochemical model to discuss the effect of our methods.

The rest of this paper is organized as follows. In Section 2, we introduce the architecture of the ReCSiP system. Section 3 describes an overview and process flow of our methods. Then, Section 4 presents the method to extract and combine common operators in data flow graphs (DFGs) of rate law functions in biochemical models. Clustering methods using similarity of rate law functions is shown in Section 5. Section 6 shows results of evaluation experiments and discusses the effectiveness of our methods. Finally Section 7 concludes this paper.

2. ReCSiP

ReCSiP organizes several Solver modules (**Fig. 1**) to solve ordinary differential equations in a reaction model, and a Switch module to connect them together on an FPGA. A Solver consists of two submodules, a Solve Core to calculate a rate law function and an Integrator for numerical integration. A Solver Core consists of several pipelined single-precision floating point arithmetic units and shift registers. The behavior of this deep arithmetic pipeline is statically scheduled.

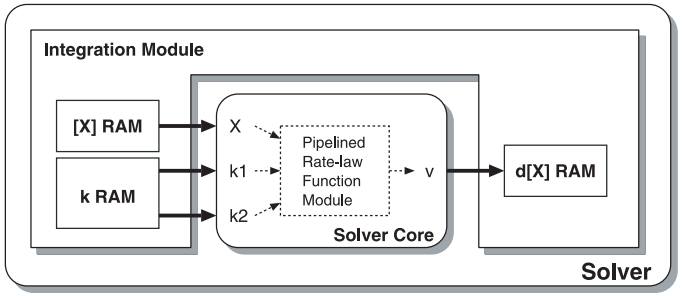


Fig. 1 Organization of Solver modules.

Calculation of a rate law function in a Solver Core takes concentration values of the substances related to the reaction and kinetic rate coefficient values as input. For these input data, a Solver Core has one input port for concentration values (X in Fig. 1) and two input ports (k1 and k2) for the rate coefficient values. On the other hand, only one output port (v) is provided to output the calculated reaction velocity. Inner structure of a Solver Core differs by the rate law function to be calculated.

Depending on the rate law function, the number of input data may exceed the number of input ports. In this case, all input data can not be fed into the Solve Core in one clock cycle. The number of clock cycles required for data input is $P = \max(N_{\mathrm{x}}, \lceil N_{\mathrm{k}}/2 \rceil)$, where N_{x} and N_{k} denote the number of concentration values and coefficient values, respectively. In case of $P = 1$, if enough number of arithmetic modules are prepared for the pipeline, the Solve Core can calculate a rate law function every clock cycle. In case of $P \geq 2$, the Solver Core can calculate individual functions every P clock cycle, resulting in idle cycles on some arithmetic modules. This offers another chance to reduce the hardware amount by sharing the some arithmetic modules among several different pipeline stages.

In ReCSiP, a Solver Core can not calculate multiple rate law functions simultaneously. However, implementation of multi-functional Solver Cores which

support multiple rate law functions and switch the function on demand is possible.

3. Overview of the Proposed Method

Our approach first makes groups of similar rate law functions of a given simulation target written in SBML [5]. Then the common subgraphs are extracted from the corresponding DFGs and are combined to reduce the circuit size.

The proposed process flow consists of the following four steps (**Fig. 2**):

(1) Extract rate law functions from a given SBML file.
(2) Convert the rate law functions into DFGs.
(3) Divide DFGs into some clusters according to their similarity.
(4) Combine common subgraphs of DFGs in each cluster.

The combined DFGs are converted into pipelined hardware of a Solver Core by performing pipeline scheduling, input data scheduling, and arithmetic resource binding [13].

For example, the rate law functions for irreversible mixed inhibition kinetics (UMI) and reversible uncompetitive inhibition (UUCR), defined in SBML level 1 are expressed as

$$v = \frac{V \times S/K_m}{1 + I/K_{is} + (S/K_m)(1 + I/K_{ic})} \quad (1)$$

and,

$$v = \frac{V_f \times S/K_{ms} - V_r \times P/K_{mp}}{1 + (S/K_{ms} + P/K_{mp})(1 + I/K_i)} \quad (2)$$

respectively. These equations are converted into DFGs as shown in **Fig. 3**. Then, the common subgraphs corresponding to the shaded regions in Fig. 3 are extracted

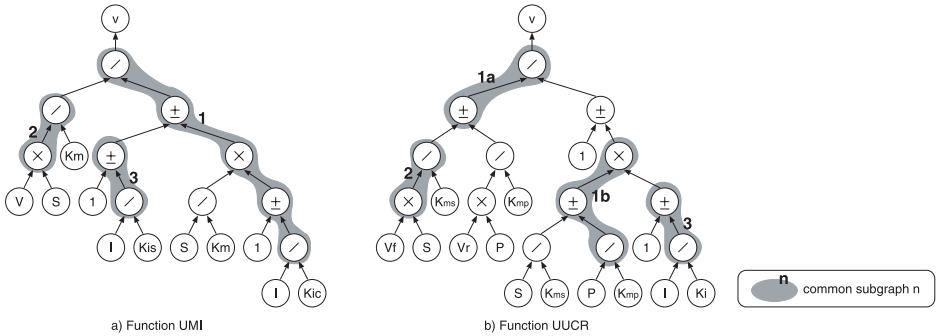


Fig. 3 Extracted common subgraphs in two DFGs.

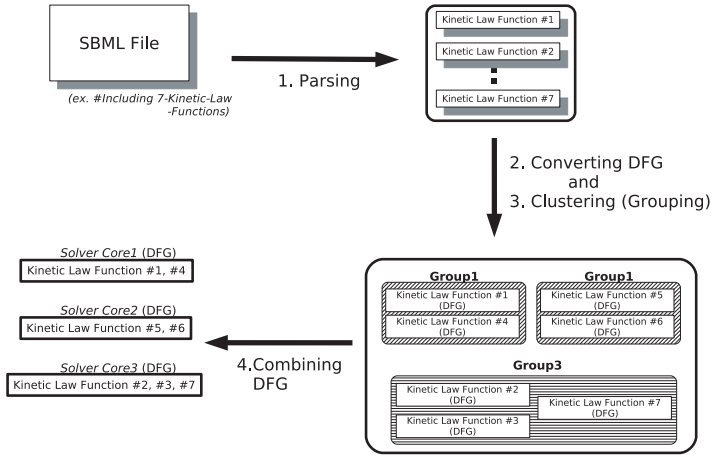


Fig. 2 Proposing process flow.

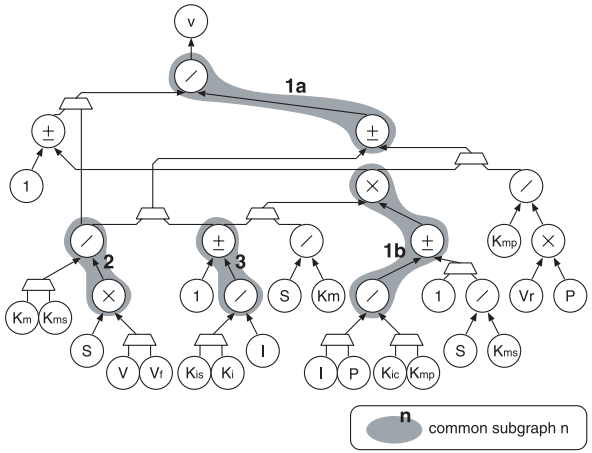


Fig. 4 Combined graph.

and the two DFGs are combined by sharing the common subgraphs. The other different parts between the DFGs are switched by extra multiplexers as shown in **Fig. 4**. When more than three DFGs are combined, the above-mentioned approach is iteratively applied in a one-by-one manner.

4. Common Subgraph Extraction

4.1 Terminologies and Definitions

A data flow graph (DFG) is a labeled directed graph $g = (V, E, l)$, where V is the set of nodes, and $E \subseteq V \times V$ is the set of edges. An ordered pair of nodes $e = (v_i, v_j) \in E$ represents g has an edge directed from v_i to v_j. Using the label map $l : V \to L_{op}$, the operation type of the node v is denoted as $l(v)$, where L_{op} is the set of operation types. In this work, L_{op} consists of the 4 elements corresponding to addition/subtraction, multiplication, division, and exponentiation, respectively.

Definition 4.1 (sharable nodes) Suppose two nodes v_1 and v_2 belong to DFGs $g_1 = (V_1, E_1, l_1)$ and $g_2 = (V_2, E_2, l_2)$, respectively. The nodes v_1 and v_2 are *sharable* if and only if $l_1(v_1) = l_2(v_2)$.

Definition 4.2 (common subgraph) Let V' be a subset of V and let E' be a subset of E such that $v_i, v_j \in V'$ for every $e = (v_i, v_j) \in E'$. Let $l' : V' \to L_{op}$ be a partial map of l such that $l'(v) = l(v)$. Then, the DFG $g' = (V', E', l')$ is called a *subgraph* of $g = (V, E, l)$. Let g_1 and g_2 be DFGs and $g'_1 = (V'_1, E'_1, l'_1)$ and $g'_2 = (V'_2, E'_2, l'_2)$ be subgraphs of g_1 and g_2, respectively. Suppose $f : V'_1 \to V'_2$ is a one-to-one correspondence between the sets of nodes such that (v_i, v_j) is an edge of g'_1 if and only if $(f(v_i), f(v_j))$ is an edge of g'_2, and v and $f(v)$ are sharable for every $v \in V'_1$. Then g'_1 and g'_2 are called *common subgraphs* of g_1 and g_2.

Definition 4.3 (connectivity) Consider an alternating sequence of nodes and edges of a DFG $g = (V, E, l)$ in the form of

$$v_0, e_0, v_1, e_2, v_2, \ldots e_{n-1}, v_{n-1}, e_n, v_n$$

where $(v_{i-1}, v_i) \in E$ or $(v_i, v_{i-1}) \in E$ for each e_i. The sequence is called a *path* if all the nodes on the sequence are distinct. A graph is said to be *connected* if there is a path between any two of its nodes. Otherwise, the graph is called *unconnected*.

The first step of combining DFGs is to extract common subgraphs among the DFGs. This paper presents and compares two approaches; extraction of unconnected common subgraphs and connected common subgraphs. In the followings, $v_i(j)$ denotes the node j in the i-th DFG g_i.

```
INPUT: Graph graph1, Graph graph2
for(Node i=0; i<total_node(graph1); i++){
  for(Node j=0; j<total_node(graph2); j++){
    Graph common_node_chunk=COMMON_GRAPH_SET(i, j, graph1, graph2);
    if(common_node_chunk.size()>THRESHOLD)
      COMBINE_LIST(common_node_chunk);
  }
}
```

Fig. 5 Extraction of unconnected common subgraph.

4.2 Searching Unconnected Common Subgraphs

The unconnected common subgraph discovery algorithm is described in **Fig. 5** and **Fig. 6**. This algorithm checks all combinations of nodes to detect common subgraphs, since DFGs of rate law functions for biochemical models are fortunately not too large to traverse all nodes and edges in a pair of DFGs. The common subgraphs that have $(v_1(i), v_2(j))$ as their root nodes are extracted from two DFGs g_1 and g_2 by calling `COMMON_GRAPH_SET` function, passing the combination of nodes $(v_1(i), v_2(j))$ as arguments. `COMBINE_LIST` function keeps the common subgraphs in the list when the number of nodes exceeds the value `THRESHOLD`. `THRESHOLD` sets the minimum grain size of common subgraphs to be candidate of combining. Smaller `THRESHOLD` results in larger number of candidate subgraphs. However, it is expected that the overhead of graph combining increases when too many small subgraphs are chosen. While there will be a right trade-off point, the value of `THRESHOLD` is set to 2 in this work, in order to evaluate the case of the finest grain size.

The `IsCommonNode` function shown in Fig. 6 decides whether the corresponding nodes are sharable (i.e., whether the nodes $v_1(i)$ and $v_2(j)$ are the same operators). The adder and subtractor are regarded to be sharable. A sharable

```
INPUT: Int i, Int j, Graph graph1, Graph graph2
queue.push(i, j);
while(!queue.isEmpty()){
  (i, j)=queue.pop();
  if(IsCommonNode(v1(i), v2(j))){
    if(list.find(i, *)==NOT_FOUND)
      if(list.find(*, j)==NOT_FOUND)
         list.push(i, j);
    for(k=0; k<total_child_node(v1(i)); k++){
      for(m=0; m<total_child_node(v2(j)); m++){
        child_i=get_child(v1(i), k);
        child_j=get_child(v2(j), m);
        queue.push(child_i, child_j);
      }
    }
  }
}
return list;
```

Fig. 6 COMMON_GRAPH_SET function for unconnected common subgraph.

combination of nodes is stored in the list, if the combination has not been stored yet. Then, the successor nodes of $v_1(i)$ and $v_2(j)$ are iteratively traversed in a breadth first search manner until reaching to the leaf nodes. As the result, the common subgraphs whose root nodes are $v_1(i)$ and $v_2(j)$ are listed in the `list`.

4.3 Searching Connected Common Subgraphs

The connected common subgraph discovery algorithm is shown in **Fig. 7** and **Fig. 8**. Figure 7 is basically the same as Fig. 5 shown in the previous subsection. The `IsCommonNode` function checks whether corresponding nodes are sharable in the same way as the algorithm in Fig. 5. The object `common_node_chunk` keeps the common subgraphs while `COMMON_GRAPH_SET` function traverses the DFGs recursively. The object `mcs` keeps the largest common subgraph extracted in the

```
INPUT: Graph graph1, Graph graph2
for(Node i=0; i<total_node(graph1); i++){
  for(Node j=0; j<total_node(graph2); j++){
    Graph common_node_chunk; common_node_chunk.clear();
    Graph tmp_mcs; tmp_mcs.clear();
    Graph mcs = COMMON_GRAPH_SET(i, j, graph1, graph2,
                                 common_node_chunk, tmp_mcs);
    if(mcs.size()>THRESHOLD)
      COMBINE_LIST(mcs);
  }
}
```

Fig. 7 Extraction of connected common subgraph.

past searches. If the number of nodes in the common subgraph stored in `mcs` is less than that of the current graph in `common_node_chunk`, `mcs` is updated.

The `IsExtend` function checks whether the current combination of the nodes can be appended in `common_node_chunk`. Here, the function returns true, only when the nodes in the common subgraph are connected.

4.4 Selection of Common Subgraphs to Combine

The common subgraphs obtained in the methods shown in the previous subsections are the candidates to be combined. In this subsection, we show how the actually combined common subgraphs are selected from these candidate common subgraphs. Since the more arithmetic units are shared, the more hardware resources can be saved, selection of candidate common subgraphs so that as many nodes as possible can be combined is desirable. That is, the objective function to be maximized by the selection algorithm described here is the number of nodes included in the selected common subgraphs. However, the two constraints below must be met to a proper construction of the pipeline:

(1) The selected common subgraph can not be overlapped.
(2) The combined graph can not have a circular dependency.

The combined graph in **Fig. 9** (c) was made from the 2 candidate common

```
INPUT: Int i, Int j, Graph graph1, Graph graph2,
       Graph common_node_chunk, Graph tmp_mcs

if(IsCommonNode(v1(i), v2(j)){
  if(IsExtend(i, j, graph1, graph2, common_node_chunk)){
     common_node_chunk.add(i, j);
     if(common_node_chunk.size()>tmp_mcs.size())
         tmp_mcs=common_node_chunk;
  }
}
for(k=0; k<total_child_node(v1(i)); k++){
     for(m=0; m<total_child_node(v2(j)); m++){
         child_i = get_child(v1(i), k);
         child_j = get_child(v2(j), m);
         tmp_mcs=COMMON_GRAPH_SET(child_i, child_j, graph1, graph2,
                                          common_node_chunk, tmp_mcs);
    }
 }
return tmp_mcs;
```

Fig. 8 COMMON_GRAPH_SET function for connected common subgraph.

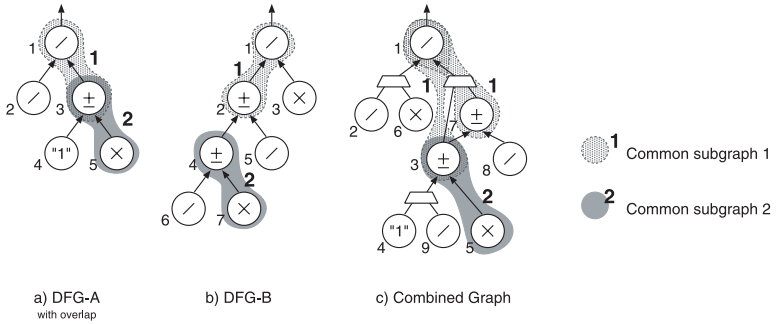


Fig. 9 DFG with overlapping.

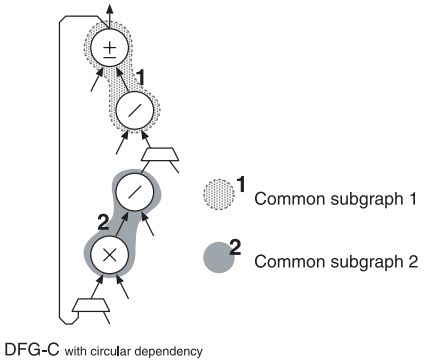


Fig. 10 Combined graph with circular dependency.

subgraphs in DFG-A (Fig. 9 (a)) and DFG-B (Fig. 9 (b)). In the combined subgraph in Fig. 9 (c), the result from node 5 is passed to node 3, then goes to node 1 via node 7 (DFG-B) or directly (DFG-A). This requires 2 adder/subtractors for Common subgraph 1 (Fig. 9), thus weakens the resource reduction efficiency. This is the reason to avoid overlaps between 2 common subgraphs.

The violation of the second constraint can make a circular dependency among common subgraphs like shown in **Fig. 10**. In this example, Common subgraph 1 and Common subgraph 2 (Fig. 10) have a mutual dependency on each other, preventing the combined hardware from being pipelined.

Combining common subgraphs to maximize the number of contained nodes is similar to knapsack problem. However, there are several differences, such as the above-mentioned constraints on the common subgraphs that can be selected at the same time, allowance of the common subgraph selection without limit of the number. Especially by the latter property, the number of combinations to search becomes enormous.

Our approach to quickly find the most effective combination of subgraphs to be combined, is use of a binary decision tree called the subgraph selection tree. **Figure 11** shows an example of the subgraph selection tree. The common subgraphs are weighted by the number of nodes, and sorted by the weight in descending order. Each common subgraph corresponds to a level of the selection tree, where

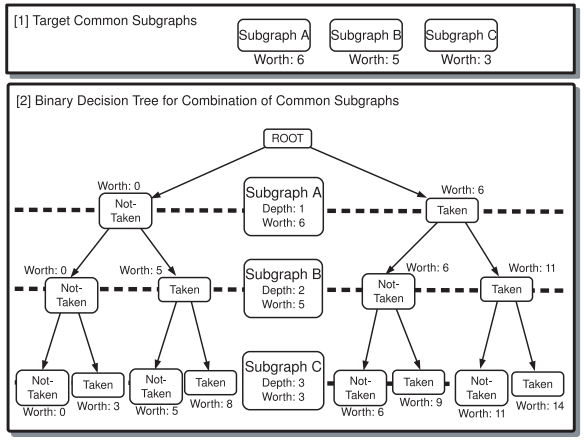


Fig. 11 Subgraph selection tree.

the root node represents the heaviest common subgraph. In Fig. 11, for example, the three common subgraphs A, B and C are in order corresponding to the levels of the selection tree from the root. The edges in the tree represent a decision to select the corresponding subgraph or not. That is, starting from the root node, the right outbound edge is followed if the Subgraph A is selected, while the left edge is followed if not. When this process reaches a leaf node, the total number of nodes included in the selected common subgraphs is obtained. The restriction constraints are represented by removing the edges corresponding to the prohibited combination of subgraphs from this tree. In addition, in order to reduce the search time we provide a threshold depth at which the search process stops. Common subgraph to combine which has the largest score in the range of this threshold can be found in this way. The effect of the threshold depth is also evaluated and discussed in Section 6.

5. Clustering DFGs

Although a single Solver Core can be generated by combining all the rate law functions included in a given simulation target, this will not be efficient in terms of performance since many multiplexers will be employed to switch the function. Instead of combining all rate law functions, making some groups of the functions and making a combined Solve Core for each group is a realistic way to achieve hardware resource reduction while alleviating the performance degradation. This paper presents and compares two clustering methods, by DFG size and by the largest common subgraph.

5.1 Clustering by DFG Size

The first method is k-means clustering [14] by the size of DFGs, intending to make groups of similar size of DFGs. Here, we use the number of nodes and the height of a DFG as a metric.

In this method, a given set of n DFGs $(g_i, i = 1, 2, \ldots, n)$ are classified into k clusters $(S_j, j = 1, 2, \ldots, k)$. The k-means clustering is to minimize:

$$V = \sum_{j=1}^{k} \sum_{g_i \in S_j} \left((n_{g_i} - n_{c_j})^2 + (h_{g_i} - h_{c_j})^2 \right) \tag{3}$$

where n_{g_i}, h_{g_i}, and $c_j \in S_j$ represent the number of nodes of DFG g_i, the height of g_i, and the center of gravity in the cluster S_j, respectively. The centers of gravity for the parameters (n_{c_j}, h_{c_j}) are defined as:

$$n_{c_j} = \frac{\sum_{g_i \in S_j} (n_{g_i})}{|S_j|} \tag{4}$$

$$h_{c_j} = \frac{\sum_{g_i \in S_j} (h_{g_i})}{|S_j|} \tag{5}$$

5.2 Clustering by the Largest Common Subgraph

The other method measures the similarity of DFGs by the size of the largest common subgraph. A similar pair of DFGs are expected to have larger common subgraph. In this method, a distance between DFGs g_i and g_j is defined as:

$$d_{ij} = 1 - \frac{1}{2} \left(\frac{\mathrm{MCS}(g_i, g_j)}{n_{g_i}} + \frac{\mathrm{MCS}(g_i, g_j)}{n_{g_j}} \right) \tag{6}$$

where $\mathrm{MCS}(g_i, g_j)$ is the total number of nodes in the largest common subgraphs between g_i and g_j. Since $\mathrm{MCS}(g_i, g_j)$ is equal to or less than n_i and n_j, d_{ij} represents similarity in the range of

$$0 \leq d_{ij} \leq 1. \tag{7}$$

The higher similarity g_i and g_j have, d_{ij} becomes smaller.

To cluster DFGs, values of d_{ij} for all combinations of given DFGs are calculated

first. Then the DFGs are clustered by linking two DFGs when the distance is lower than a threshold value. The number of clusters can be adjusted by the threshold value.

6. Evaluation and Discussion

In this section, the proposed methods and their implementation alternatives are evaluated and compared mainly in the following points of view:

(1) Common subgraph extraction: unconnected or connected.
(2) Subgraph selection tree: the threshold depth.
(3) Clustering: k-means or largest common subgraph.

The proposed methods were implemented in C++ (gcc4.1.2 +O3) with Boost C++ library and libSBML [15], and were evaluated using 32 SBML predefined rate law functions [6]. Pipeline scheduling and arithmetic resource sharing were done by list scheduling policy [7),13)]. This C++ program reads a model described in an SBML file, then generates Verilog-HDL modules required to solve the model. Generated pipelines were synthesized, placed and routed on a Xilinx VirtexII-Pro XC2VP70-5 on the ReCSiP board, using a Slang SSE 10.1 tool. **Table 1** shows the number of slices for arithmetic units and multiplexers used in the evaluation.

6.1 Connectivity of Combine Candidate Common Subgraphs

In order to reveal influence of common subgraph connectivity independently on clustering methods, randomly selected sets of 3, 5, and 10 rate law functions from 32 SBML predefined functions are combined Solver Cores with a threshold depth of 10.

Table 1 Slice counts for arithmetic units and multiplexers.

operation module	slices
addition/subtraction	605
multiplication	230
division	1,684
exponentiation	5,406
2-input multiplexer	19
3-input multiplexer	37
4-input multiplexer	82
5-input multiplexer	55
6-input multiplexer	92
7-input multiplexer	144
8-input multiplexer	180

Figure 12 shows hardware amount (FPGA slice count) for combined Solver Cores on average of 20 trials of the abovementioned random selections. The bars show the total slice counts of combined modules, and the lines show relative slice usage compared to uncombined design. For example, the 10-function Solver Core using connected subgraphs only takes 41.99% compared to the total slices required for 10 single function Solver Cores. Figure 12 also gives a breakdown of total FPGA slices for arithmetic units and shift registers. For 5-function Solver Cores and 10-function Solver Cores, the method using connected common subgraphs shows better results than extracting unconnected common subgraphs. On the other hand, use of unconnected common subgraphs slightly saved FPGA slices for 3-function Solver Cores.

Because an unconnected common subgraph tends to have larger number of nodes compared to connected ones, more arithmetic units were combined for the 3-function Solver Cores. However, large common subgraphs tend to easily violate the restriction constraints described in Section 4. This is why the hardware reduction effect was diminished according to increase of the number of combined functions.

Figure 12 shows the relative size of the combined Solver Cores normalized to the total size of the original, uncombined single function Solver Cores. A

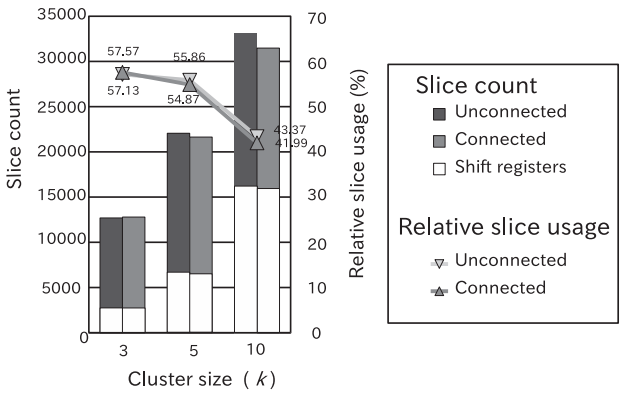


Fig. 12 Total slice usage and fraction of shift registers.

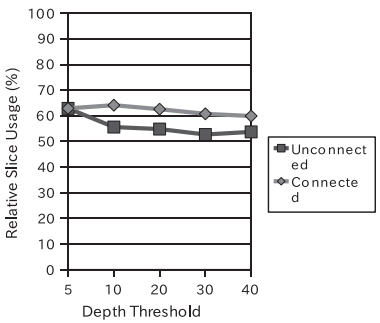


Fig. 13 3-function Solver Cores.

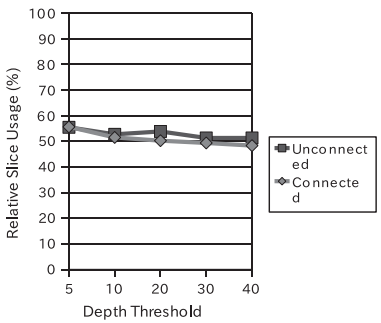


Fig. 14 5-function Solver Cores.

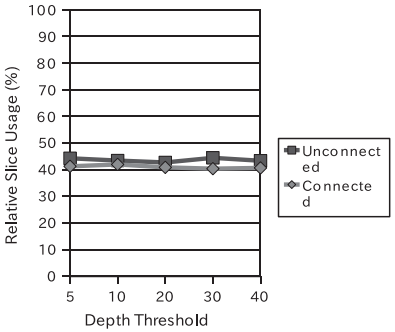


Fig. 15 10-function Solver Cores.

10-function Solver Core requires only 40% of hardware resources compared to when 10 single function Solver Cores are separately implemented. It is also suggested the overhead of shift registers would become unignorable to increase of the number of combined functions.

6.2 Threshold Depth on the Subgraph Selection Tree

To evaluate influence of the threshold depth in the selection tree, the same sets of rate law functions as used in the previous subsection were combined, changing the threshold depth in 5, 10, 20, 30 and 40. **Figures 13**, **14**, **15** summarize the relationship between the threshold depth and hardware amount of generated Solver Cores. These data are averaged values of 20 sets.

As shown in these results, the effect of the threshold depth saturates around 20 to 30. This is because deeper search increases the risk of constraint violations on subgraph selection. For the 5-function Solver Cores and 10-function Solver Cores, the method with connected common subgraphs showed better results than the method with unconnected subgraphs for every threshold depth. This tells the advantage of connected subgraphs is not degraded by the threshold depth. The average numbers of the candidate common subgraphs found over the 20 trials were, 45.95, 88.15, and 304.45 for the 3-function, 5-function, and 10-function combine, respectively.

6.3 Clustering Method

To evaluate and compare the two clustering methods described in Section 5, we classified the 32 SBML predefined functions into 5 clusters with the both methods. **Table 2** shows two negative effects of the combine that is, deterioration rate of latency and frequency against the single function Solver Cores in each group, as well as the merit of the method in the relative number of required FPGA slices. Pipeline latency extends to fit to the longest pipeline in the combined group. Frequency degradation comes from the longer wires in the larger combined modules.

Table 2 Comparison of clustering methods (%).

	k-means	Largest common graph
Latency degradation rate (%)	19.16	10.87
Frequency degradation rate (%)	8.33	7.43
Relative slice counts (%)	51.11	39.27

In terms of both of the hardware reduction and the performance degradation, the clustering method using the largest common subgraph presented better results compared to the other method as shown in Table 2. This result supports the importance of considering not only size of DFGs but also similarity.

6.4 Evaluation with Practical Model Simulation

To verify the practicality of these proposed methods, the *Drosophila* circadian rhythm model by Leloup, et al. [16] was used as a benchmark. The model mathematically represents the cyclical alteration in expression of PER protein and TIM protein which are responsible for circadian rhythm of *Drosophila* by a negative feedback mechanism with PER-TIM complex. This model consists of 24 reactions and 7 rate law functions.

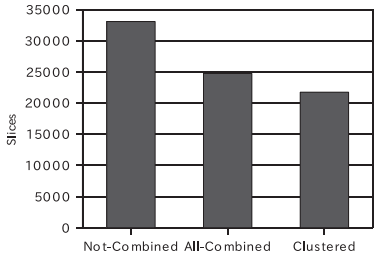


Fig. 16 Slices usage for three implementations.

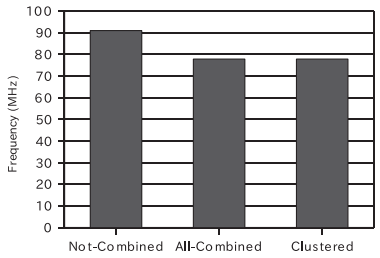


Fig. 17 Maximum frequency for three implementations.

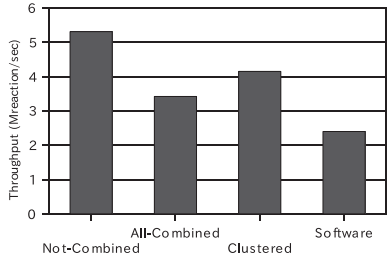


Fig. 18 Simulation throughput for three implementations and software implementation.

For comparison, we prepared the following three types of implementation alternatives:

(1) Not-Combined: the seven rate law functions were implemented with seven individual Solver Cores without combining.

(2) All-Combined: all the seven rate law functions were combined into one 7-function Solver Core.

(3) Clustered: the seven rate law functions were clustered into 3 groups to generate three multi-function Solver Cores. The number of groups was chosen to put 2 or 3 functions to each group.

For All-Combined and Clustered implementations, connected common subgraphs were extracted and they were searched to the threshold depth of 30 and the functions were clustered by using their largest common subgraphs.

Figure 16 shows the number of FPGA slices required for each implementation. The Clustered implementation achieved the best hardware reduction effect. Although All-Combined implementation shared the largest number of nodes in a DFG level, the Clustered implementation was the best since costly arithmetic units such as exponentiation units were not effectively shared in pipeline scheduling. This means that effect of the combining method is influenced not only how many nodes are shared in a DFG level but also arithmetic pipeline scheduling that is a future topic beyond this work. Total size of Not-Combined and All-Combined implementations had shrunken to 74.82% and 65.73% of the original hardware resources, respectively.

Figure 17 shows the maximum frequency of these three implementations. Because extra multiplexer for switching functions were inserted when DFGs were combined, both of All-Combined and Clustered implementations showed 15.38% of frequency degradation against the Not-Combined implementation. **Figure 18** shows simulation throughput for the three implementations and software implementation of the same simulation which runs on Linux PC equipped with a Pentium 4 CPU. In this evaluation, the maximum frequency of the FPGA was limited to 33 MHz due to a clock generator implemented on the ReCSiP board. Compared to the software result, Not-Combined, All-Combined, and Clustered implementation achieved 2.21 times, 1.43 times, and 1.73 times throughput improvements, respectively. In addition, if these implementations run with the frequency shown in Fig. 17, Not-Combined, All-Combined and Clustered implementation would achieve 6.1 times, 3.33 times, and 4.08 times speedup to software, respectively. Although our hardware reduction method with function combining brings on approximately 20% of performance overhead, large hardware reduction effects are achieved with a proper clustering method. Since the performance advantage to software execution is also retained, we conclude that the method is an effective technique especially when large scale biochemical models are simulated.

7. Conclusion

This paper has shown a systematic method to cope with FPGA resource issues in ReCSiP simulation. The proposed method reduces hardware resources by finding and combining common subgraphs among DFGs for rate law functions in a biochemical model, so that cost-effective multi-functional Solver Cores to be

generated. We also presented two clustering methods to alleviate the negative effects on performance for combined Solver Cores. As a result of evaluation with a practical biochemical model, it was shown that our method enabled the simulation with 66% of the original hardware amount, at a reasonable cost of 20% performance overhead.

Our future work includes addressing for further efficient DFG clustering methods, combining techniques unified with arithmetic pipeline scheduling, and hardware reduction techniques for shift registers for combining larger number of functions.

Acknowledgments The authors would like to express their sincere gratitude to Prof. Sueyoshi and Prof. Harasawa of Nagasaki University for their fruitful suggestion and discussion. This work was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B). This work was also supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Cadence Design Systems, Inc.

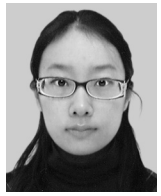
References

1) Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter, J., et al.: E-CELL: Software environment for whole-cell simulation, *Bioinformatics*, Vol.15, No.1, p.72 (1999).
2) Moraru, I., Schaff, J., Slepchenko, B. and Loew, L.: The virtual cell: An integrated modeling environment for experimental and computational cell biology., *Annals of the new york academy of sciences*, Vol.971, p.595 (2002).
3) Osana, Y., Yoshimi, M., Iwaoka, Y., Kojima, T., Nishikawa, Y., Funahashi, A., Hiroi, N., Shibata, Y., Iwanaga, N., Kitano, H., et al.: ReCSiP: An FPGA-based general-purpose biochemical simulator, *Electronics and Communications in Japan Part 2 Electronics*, Vol.90, No.7, p.1 (2007).
4) Osana, Y., Yoshimi, M., Iwaoka, Y., Kojima, T., Nishikawa, Y., Funahashi, A., Hiroi, N., Shibata, Y., Iwanaga, N., Kitano, N., et al.: Performance Evaluation of an FPGA-based Biochemical Simulator ReCSiP, *Proc. FPL*, pp.845–850 (2006).
5) Hucka, M., Finney, A., Bornstein, B., Keating, S., Shapiro, B., Matthews, J., Kovitz, B., Schilstra, M., Funahashi, A., Doyle, J., et al.: Evolving a lingua franca and associated software infrastructure for computational systems biology: The Systems Biology Markup Language (SBML) project, *Syst. Biol*, Vol.1, No.1, p.41 (2004).
6) Hucka, M., Finney, A., Sauro, H. and Bolouri, H.: *Systems Biology Markup Language (SBML) Level 1: Structures and Facilities for Basic Model Definitions*, MC 107-81 California Institute of Technology, Pasadena, CA 91125, USA (2003).
7) Iwanaga, N., Shibata, Y., Yoshimi, M., Osana, Y., Iwaoka, Y., Fukushima, T., Amano, H., Funahashi, A., Hiroi, N., Kitano, H., et al.: Efficient Scheduling of Rate Law Functions for ODE-based Multimodel Biochemical Simulation on an FPGA, *Proc. FPL*, pp.666–669 (2005).
8) Yamada, H., Iwanaga, N., Shibata, Y., Osana, Y., Yoshimi, M., Iwaoka, Y., Nishikawa, Y., Kojima, T., Amano, H., Funahashi, A., et al.: A Combining Technique of Rate Law Functions for a Cost-Effective Reconfigurable Biological Simulator, *International Conference on Field Programmable Logic and Applications, 2007. FPL 2007*, pp.808–811 (2007).
9) DeSouza, C., Lima, A., Moreano, N. and Araujo, G.: The datapath merging problem in reconfigurable systems: Lower bounds and heuristic evaluation, *Lecture notes in computer science*, Vol.3059, pp.545–558 (2004).
10) Smith, A., Constantinides, G. and Cheung, P.: Fused-Arithmetic Unit Generation for Reconfigurable Devices using Common Subgraph Extraction, *International Conference on Field-Programmable Technology, 2007. ICFPT 2007*, pp.105–112 (2007).
11) Moreano, N., Borin, E., DeSouza, C. and Araujo, G.: Efficient datapath merging for partially reconfigurable architectures, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.24, No.7, pp.969–980 (2005).
12) Fazlali, M., Fallah, M., Zolghadr, M., Zakerolhosseini, A. and Tehran, I.: A New Datapath Merging Method for Reconfigurable System, *Proceedings of the 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications*, Springer, pp.157–168 (2009).
13) Ishimori, T., Yamada, H., Shibata, Y., Osana, Y., Yoshimi, M., Nishikawa, Y., Amano, H., Funahashi, A., Hiroi, N. and Oguri, K.: Pipeline Scheduling with Input Port Constraints for an FPGA-Based Biochemical Simulator, *Proceedings of the 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications*, Springer, pp.368–373 (2009).
14) MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations (1966).
15) Bornstein, B., Keating, S., Jouraku, A. and Hucka, M.: LibSBML: An API Library for SBML, *Bioinformatics*, Vol.24, No.6, p.880 (2008).
16) Leloup, J. and Goldbeter, A.: Chaos and birhythmicity in a model for circadian oscillations of the PER and TIM proteins in Drosophila, *Journal of theoretical biology*, Vol.198, No.3, pp.445–459 (1999).

(Received December 1, 2009)
(Revised February 26, 2010)
(Accepted April 14, 2010)
(Released August 16, 2010)

(Recommended by Associate Editor: *Hiroshi Saito*)

Hideki Yamada received his B.E. and M.E. degrees in information and computer sciences from Nagasaki University in 2007 and 2009, respectively. He currently works for Toshiba Corporation. His research interests include reconfigurable systems, scheduling and compilers.



Yui Ogawa received her B.E. degree in information and computer sciences from Nagasaki University in 2010. She is currently a M.E. student at Nagasaki University. Her research interests include reconfigurable systems.

Tomonori Ooya received his B.E. degree in information and computer sciences from Nagasaki University in 2009. He is currently a M.E. candidate at Nagasaki University. His research interests include reconfigurable systems and parallel processing.

Tomoya Ishimori received his B.E. and M.E. degrees in information and computer sciences from Nagasaki University in 2008 and 2010, respectively. He currently works for Namura Information Systems Co., Ltd. His research interests include arithmetic scheduling for reconfigurable pipeline systems.

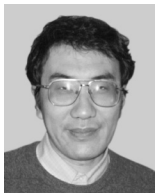
Yasunori Osana received his Ph.D. degree from the Department of Information and Computer Science of Keio University in Yokohama, Japan in 2006. He is currently an assistant professor in the Department of Computer and Information Science, Seikei University. His research interests include computer architecture, reconfigurable systems and their application in scientific computing.

Masato Yoshimi received his B.E., M.E. and Ph.D. degrees from Keio University, Japan, in 2004, 2006 and 2009. He is currently an Assistant Professor at Doshisha University, Japan. His research interests include the area of reconfigurable computings, parallel processings and intelligent systems.

Yuri Nishikawa received her B.E. and M.E. degrees from Keio University, Japan, in 2006 and 2008. She is currently a Ph.D. candidate at Keio University. Her research interests include the areas of high performance computing, reconfigurable computing and interconnection networks.

Akira Funahashi received his B.E. degree in electrical engineering and M.E. and Ph.D. degrees in computer science from Keio University, Japan, in 1995, 1997 and 2000, respectively. His research interests include the areas of systems biology, computational biology, interconnection networks, and parallel processing. He was a Research Fellow with the Japan Society of the Promotion of Science (DC1) from 1997 to 2000 and a Research Associate with the Department of Information Technology, Mie University, Japan, from 2000 to 2002. He then joined the Kitano Symbiotic Systems Project, JST, and the Systems Biology Institute as a Researcher before joining Keio University in 2007. He is now an Associate Professor in the Department of Biosciences and Informatics, Keio University.

Noriko Hiroi received her PharB and PharM degrees from Tokyo University of Science, and her Ph.D. degree in Medical Science from the University of Tokyo in 1996, 1998 and 2002, respectively. Between 2002 and March 2007, she joined the Kitano Symbiotic Systems Project, ERATO-SORST, JST as a postdoctoral researcher. From 2007 to 2009 she was a postdoctoral fellow at EMBL-EBI, UK. She is now a Research Associate in the Department of Biosciences and Informatics, Keio University. Her research interests include the areas of systems biology, computational biology, biological molecular networks, and quantitative biology.



Hideharu Amano received his Ph.D. degree from Keio University, Japan in 1986. He is now a Professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable computing.

Yuichiro Shibata received his B.E. degree in electrical engineering, M.E. and Ph.D. degrees in computer science from Keio University, Japan, in 1996, 1998 and 2001, respectively. Currently, he is an Associate Professor at the Department of Computer and Information Sciences, Nagasaki University. He was a Visiting Scholar at University of South Carolina in 2006. His research interests include reconfigurable systems and parallel processing. He won the Best Paper Award of IEICE in 2004.

Kiyoshi Oguri received his B.S. and M.S. degrees in physics from Kyushu University, Japan, in 1974 and 1976, respectively. He also received his Ph.D. degree in information engineering from the same university in 1997. Since joining NTT in 1976, he have been engaged in the research, design and development of high-end general purpose computer, high-level logic synthesis system and a wired logic-based dynamic computing architecture. Currently, he is a Professor of Nagasaki University, Japan. His research interests are in hardware modeling, high-level synthesis, FPGA-related systems, and Plastic Cell Architecture. Prof. Oguri received the Motooka Prize in 1987, the Best Paper Award of IPSJ in 1990, the Okochi Memorial Technology Prize in 1992, the Achievement Award of IEICE in 2000, and the ACM Gordon Bell Prize in 2009.