A COMBINING TECHNIQUE OF RATE LAW FUNCTIONS FOR A COST-EFFECTIVE RECONFIGURABLE BIOLOGICAL SIMULATOR

Hideki Yamada, Naoki Iwanaga
Yuichiro Shibata
Nagasaki University,
Nagasaki, Japan

Yasunori Osana, Masato Yoshimi, Yow Iwaoka
Yuri Nishikawa, Toshinori Kojima
Hideharu Amano, Akira Funahashi
Keio University, Yokohama, Japan

Noriko Hiroi
EMBL-EBI Janet Thornton Group,
Cambridge, United Kingdom

Hiroaki Kitano
Kitano Symbiotic Systems Project,
ERATO-SORST, JST
Tokyo, Japan

Kiyoshi Oguri
Nagasaki University
Nagasaki, Japan

ABSTRACT

In order to simulate large scale biological models with a reconfigurable FPGA-based biochemical simulator system, reduction of required resources are essential. This paper proposes a method which combines common terms in rate law functions appeared in biochemical models and generates a shared hardware module used for numerical integration. In this approach, two functions are combined in a tree structure level, followed by pipeline scheduling and arithmetic module binding. The evaluation result reveals that this approach reduces hardware resources by 31.4 % on average at the cost of 14.4 % throughput degradation.

1. INTRODUCTION

Mathematical modeling and simulation of biological processes are now essential to understand cellular activity in a system level based on knowledge accumulated in wet experiments. While a lot of biochemical simulators such as E-Cell [1] and Virtual Cell [2] have been developed, it takes a long time and requires large computational resources to simulate practical models. To cope with this problem, an FPGA-based biochemical simulator, ReCSiP has been developed [3][4]. ReCSiP features deep pipelined hardware modules to compute reaction rates that are automatically generated from given model description [5]. However, since biochemical models contain various varieties of reaction rate functions, these modules require a large FPGA area and tend to restrict extraction of reaction-level parallelism.

This paper shows a novel method of automated design of hardware modules to compute reaction rates focusing on hardware resource optimization. In this approach, rate raw functions in model description are first converted into tree structure and common sub-trees are found and combined. Then, pipelined hardware is synthesized from the combined tree structure through arithmetic scheduling.

2. RECSIP

ReCSiP hardware is a PCI board which has a Xilinx XC2VP70 and 8 chips of 18Mbit QDR-I SRAM. Simulator on the FPGA consists of Solver modules which calculate change in concentration of biochemical substances solving differential equations of reactions. Multiple Solver modules are allocated in the FPGA and they are connected each other by communication switch.

As illustrated in Fig.1, a Solver consists of two modules, an Integrator and a Solver Core. While the Solver Core calculates velocity of a reaction, the Integrator performs numerical integration using the Solver Core. Although the Solver Core requires a lot of floating point arithmetic operations, high degree of throughput can be achieved by statically and completely scheduling the pipeline structure. A Solver Core is reaction-specific hardware that takes concentrations of substances and coefficients from three input ports (X for concentrations and the others for coefficients) and outputs the velocity. Biologists often want to launch multiple simulations on the same target with different parameters, and this pipelined structure is quite efficient to extract thread-level parallelism. On the other hand, by alleviating hardware costs of Solver Cores, it is possible to devote the rest of FPGA area to additional Solver Cores for parallel execution.

3. SOLVER CORE COMBINING ALGORITHM

Systems Biology Markup Language (SBML), which is an XML-based language commonly used for biological modeling, defines 33 frequently used rate law functions as *predefined functions* [6]. Our early work revealed that some of the predefined functions have similar structure and they can be combined into a single Solver Core by sharing common

1-4244-1060-6/07/$25.00 ©2007 IEEE.

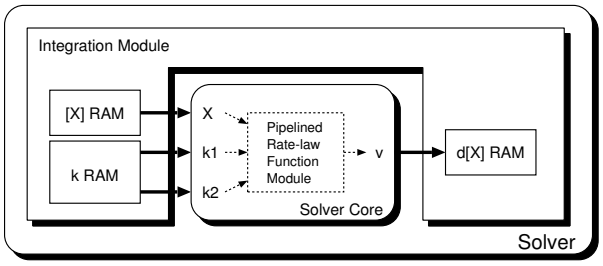

Fig. 1. Solver module

```
1 . int Score(TreeNode *p1, TreeNode *p2){
2 .  if(is_equal(p1, p2){
3 .   int score=1;
4 .   for(unsigned int id=0;
5 .       id < p1->getNumChildren() &&
6 .       id < p2->getNumChildren() ; id++){
7 .     score+=Score(p1->getChild(id),
8 .                  p2->getChild(id));
9 .   }
10.   return score;
11.  }
12.  return 0;
13. }
```

Fig. 2. Calculation of a coincidence score

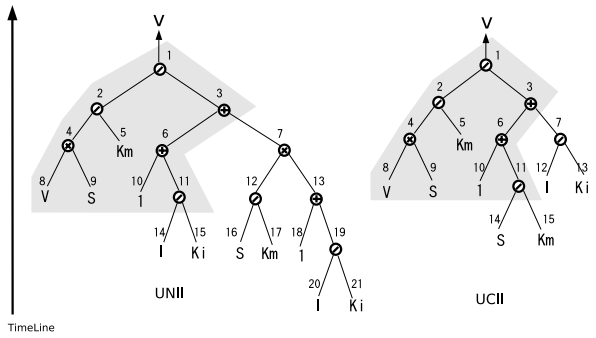

Fig. 3. Common subtrees with the highest coincidence score

arithmetic modules [5]. In this paper, a generic combining method that can support any rate law functions including non-predefined functions is proposed.

While part of traditional technique of High Level Synthesis(HLS) scheduling methods [7] [8], can be applied for designing of pipelined Solver Core modules, this paper focuses on combining of rate law functions and its effect on hardware costs. Resemblance of trees is commonly calculated by sophisticated algorithms such as Tree Kernel [9] [10]. However, size of tree for typical rate raw functions is small. Therefore, we adopt a simple approach when comparing trees as described below.

3.1. Combining of Trees

The method shown here first combines two given rate law functions in a tree structure level, and then converts to a combined Solver Core through pipeline scheduling and arithmetic resource binding. The tree combining process consists of 3 steps as follows.

1. Given rate law functions are individually converted to tree structures.
2. For every two-node combination between the two trees, a coincidence score, which represents how large subtrees of the same topology the two nodes have, is calculated by comparing two subtrees connected to the two nodes.
3. The two trees are combined by sharing the common subtree that marks the highest coincidence score.

For example, the predefined rate law functions UNII and UCII are defined as $v = \frac{VS/K_m}{1+I/K_i+(S/K_m)(1+I/K_i)}$ and $v = \frac{VS/K_m}{1+S/K_m+I/K_i}$, respectively. In Step 2, a coincidence score is calculated according to the list shown in Fig. 2 for every pair of nodes between the given two trees. The arguments of the `Score()` function are pointers to the nodes. A score value of 1 is given if these nodes represent the same operator or the same input variable. Otherwise, a score value of 0 is returned. Then, the `Score()` function is recursively called accumulating the score value until a different pair of nodes is detected.

By calling this function for all the combination of nodes, the largest common subtree is found on the pair of the highest coincidence score. This procedure can be simplified by reusing coincidence scores of subtrees. In the example of Fig. 3, the highest coincidence score is marked by the pair of the root nodes of the trees and the largest common subtrees are found as show. Then in Step 3, the original two trees are combined by sharing the largest common subtree. Different part of the trees is switched by extra multiplexers as shown in Fig. 4.

3.2. Pipeline Scheduling

Next, a Solver Core with pipelined structure is generated from the combined tree structure. At first, start times of

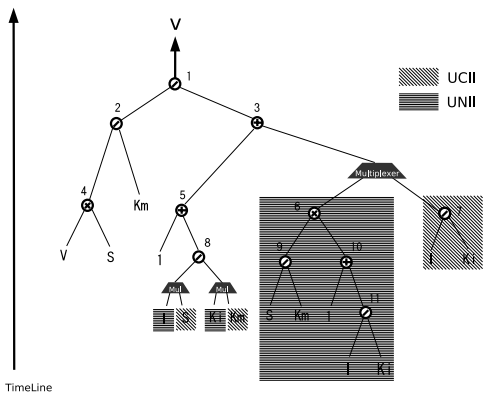

Fig. 4. Combined tree of UNII and UCII

arithmetic operators in the combined tree are temporally scheduled with the As Late As Possible (ALAP) scheduling algorithm. When the critical paths of original two trees have different length, they are adjusted to the longer one. However, as mentioned in Section 2, a bandwidth of data input to a Solver Core is limited to three input ports. Therefore, sometimes ALAP scheduling can not be possible due to lack of the input bandwidth. In this case, order of data input has to be considered since it can affect performance. Basically, data required by the operators that have earlier start time are input first. Then, priority is given to the data that are fed to operators in the combined subtree. After input order is decided, the final scheduling of arithmetic operators is fixed, modifying ALAP start time of some operators.

Our final process here is simple optimization in arithmetic resource binding. The same arithmetic operators used in different states can be shared in the same arithmetic module. When multiple operators of the same arithmetic are used in the same state, multiple arithmetic modules are generated.

4. EVALUATION AND DISCUSSION

To evaluate effect of the combining method, the algorithm was implemented in C++ (gcc 3.4.6) and was applied on the 14 rate law functions out of the 33 SBML predefined functions. Generated Verilog files are mapped on XC2VP70-6FF1517C, which is equipped on the ReCSiP-2 board, using the Xilinx ISE 8.2i tool. The frequency (MHz), latency (clock cycles), execution time (ns), throughput (Mega reactions per second) and the number of required slices of non-combined Solver Cores for these 14 functions are summarized in Table 1. The bottom three lines show the distribution (the best, worst, and average data) for each evaluated item.

4.1. Performance Comparison

The number of two-function combination from these 14 functions is $\binom{14}{2} = 91$. Table 2 shows the distribution of the implementation results for the 91 combined Solver Cores. The best frequency of 136.6 MHz is achieved by combined functions including the pair of UMAI and UNII. Meanwhile, the pair of UMAI and UCIR shows the worst frequency of 114.5 MHz. Compared to the results for non-combined Solver Cores in Table 1, the combined Solver Cores show lower average frequency, which comes from extra multiplexers to switch the combined functions.

The latency for the combined Solver Cores is distributed over the range of 64 (UAII and UCII) to 80 (UUCI and UCIR) clock cycles, also showing worse average than the non-combined Cores. When Solver Cores that have different latencies are combined, the shorter one must be adjusted to the other to have the same latency resulting in performance degradation. The trend of execution time of the combined Solver Cores is also degraded. While the best combination of UAII and UCII achieves 468.6 ns which is the same as that of the fastest non-combined UCII Core, the average execution time is decreased by 6.7%.

Table 1. Implementation results (non-combined)

Rate Law Function	Freq. (MHz)	Latency (clock)	Exec. Time(ns)	Throughput (Mreact./s)	Slices
UCTR	128.3	71	553.7	42.7	4417
UMAR	128.0	71	554.5	42.7	4419
UMR	128.1	71	554.4	42.7	4416
UNIR	125.6	71	565.4	41.9	4415
UCTI	136.6	69	505.2	68.3	4188
UMAI	136.6	69	505.2	68.3	4169
UMI	136.6	69	505.2	68.3	4178
UNII	136.6	69	505.2	68.2	4188
UUCI	136.6	69	505.2	68.3	3191
UUCR	132.0	71	537.8	44.0	4391
UAII	136.6	64	468.6	68.3	2650
UAR	129.8	69	531.6	43.3	4366
UCII	136.6	64	468.6	68.3	2650
UCIR	129.8	69	531.6	43.3	4366
Best	136.6	64	468.6	68.3	2650
Worst	125.6	71	565.4	41.9	4419
Average	132.7	69	520.9	55.6	4000

Table 2. Implementation results (combined)

	Best	Worst	Average
Frequency (MHz)	136.6	114.5	127.2
Latency (clock)	64	80	71
Execution Time (ns)	468.6	622.5	556.0
Throughput (Mreact./s)	68.3	38.2	47.6
Slices	2711	6624	5437

Throughput of Solver Cores is related to frequency and pipeline pitch which is an interval of calculations for consecutive reactions. Since pipeline pitch reflects the number of arguments to the corresponding function, throughput is affected when combined functions have the different numbers of arguments. Throughput of the combined Solver Cores is degraded by 14.4% on average.

4.2. Hardware Costs

To evaluate the influence of the function combining upon hardware costs, we use the metric called the resource reduction ratio defined as: $\left(1 - \frac{S_{A,B}}{S_A + S_B}\right) \times 100$, where S_A, S_B, and $S_{A,B}$ are the number of slices for the function A, B, and the combined function of A and B, respectively. Fig.5 shows distribution of the resource reduction ratios achieved by the 91 combined functions, in which the combinations are sorted by the resource reduction ratio on the horizontal axis. While the highest resource reduction ratio stands at 49.8% (UMR and UUCR) indicating the combining cuts off almost the half of hardware costs, the worst ratio is -6.2% (UUCI and UAII) which means increase in hardware. The average resource reduction ratio is 31.4%.

To analyze and discuss these results in detail, we focus on how many arithmetic modules are shared by the function combining in both of the tree structure level and the pipeline structure level. Table 3 summaries usage of three arithmetic modules and their reduction ratio for the best pair (UMR and UUCR), the 35th pair (UAII and UAR), and the worst

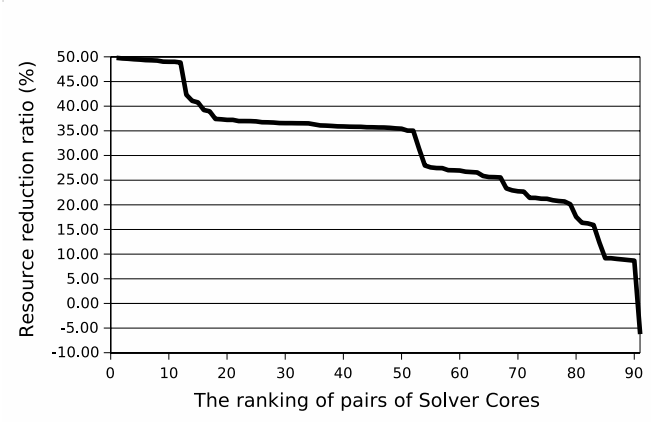

Fig. 5. Distribution of resource reduction ratios

Table 3. The number of arithmetic modules required to combined Solver Cores

		Tree structure level			Pipeline structure level		
		Non-Combined	Combined	Reduction (%)	Non-Combined	Combined	Reduction (%)
Best pair (UMR & UUCR)	adders	9	5	44.4	4	2	50.0
	multipliers	6	3	50.0	5	2	60.0
	dividers	13	7	46.1	6	3	50.0
	sum	28	15	46.4	15	7	53.3
35th pair (UAII & UAR)	adders	6	4	33.3	3	2	33.3
	multipliers	3	3	0.0	3	2	33.3
	dividers	10	7	30.0	5	3	40.0
	sum	19	14	26.3	11	7	36.3
Worst pair (UUCI & UAII)	adders	4	3	25.0	3	3	0.0
	multipliers	3	2	33.3	3	1	66.7
	dividers	8	6	25.0	4	5	-20.0
	sum	15	11	26.7	10	9	10.0

pair (UUCI and UAII). Reduction ratios of the arithmetic modules in the pipeline structure level do not directly reflect those in the tree structure level due to the pipeline scheduling and optimization described in Section 3.2.

For instance, the best pair of the original UMR and UUCR Solver Cores requires, in all, 28 arithmetic modules in the tree structure level, and our combining method reduces them to 15 modules, that is, the reduction ratio of 46.4%. Meanwhile, only 15 arithmetic modules are required when the original two Solver Cores are individually pipelined since some arithmetic modules are shared. Moreover, the combined Solver Core requires 7 arithmetic modules after pipeline scheduling, showing 53.3% of the reduction ratio.

Focusing on the 35th pair, 26.3% of the module reduction ratio is achieved in the tree structure level, and this is increased to 36.3% in the pipeline structure level. Although Solver Cores do not consist only of arithmetic modules, the reduction ratio of them is approximately consistent with the final resource reduction ratio in this case.

The pair of UUCI and UAII, which shows the worst result, achieves the nearly same module reduction ratio in the tree structure level. However, in the pipeline structure level, the module reduction ratio stands at only 10.0%, where 10 arithmetic modules are reduced to 9. Breaking down the change of the arithmetic modules, the three multipliers are shared to one by the combining, but the number of the divider is increased. Getting the extra divider while reducing 2 multipliers leads to -6.2% of the final resource reduction ratio. These results suggest that resource reduction by combining Solver Cores is much broader than a matter of node sharing in tree structured rate law functions.

5. CONCLUSION AND FUTURE WORK

This paper presented a method of automate design of a Solver Core module that computes biochemical reaction rates featuring sharing mechanism of common terms in given rate law functions. In this approach, two Solver Cores are combined in a tree structure level, followed by pipeline scheduling and arithmetic module binding.

Although only ALAP pipeline scheduling is discussed in this paper, we will evaluate the effect when more advanced scheduling algorithms [5] are applied with the combining technique. In addition, evaluation of simulation performance using actual biochemical models and combining of three or more trees are also our future work.

6. REFERENCES

[1] M. Tomita et al, "E-CELL: software environment for whole cell simulation," *Bioinformatics*, vol. 15, no. 1, pp. 72–84, 1999.

[2] I. Moraru et al, "The Virtual Cell: an Integrated modeling environment for experimental and computational Cell biology," *Annals of the New York Academy of Sciences*, vol. 971, pp. 595–596, 2002.

[3] Y. Osana et al, "An FPGA-Based, Multi-model Simulation Method for Biochemical Systems," *Proc. IPDPS*, 2005.

[4] Y. Osana et al, "Performance Evaluation of an FPGA-based Biochemical Simulator ReCSiP," *Proc. FPL*, pp. 845–850, 2006.

[5] N. Iwanaga et al, "Efficient Scheduling of Rate Law Functions for ODE-Based Multimodel Biochemical Simulation on an FPGA," *Proc. FPL*, pp. 666–669, 2005.

[6] M. Hucka et al, "Evolving a lingua franca and associated software infrastructure for computational systems biology: The systems biology markup language (SBML) project," *IEE Systems Biology*, vol. 1, no. 1, pp. 41–53, 2004.

[7] P. G. Paulin et al, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Trans. on CAD*, vol. 8, no. 9, 1989.

[8] S. Gao et al, "Pipeline scheduling for array based reconfigurable archiectures considering interconnect delays," *Proc. FPT*, pp. 137–144, 2005.

[9] M. Collins and N. Duffy, "Convolution kernels for natural language," *Advances in NIPS*, 2002.

[10] H. Kashima and T. Koyanagi, "Kernels for Semi-Structured Data," *Proc. ICML*, pp. 291–298, 2002.