RC-012

FPGAに基づく生化学シミュレータにおける反応速度式の類似性に着目したパイプライン自動構築

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

山田　英樹[†]　石森　智也[†]　大屋　智範[†]　柴田裕一郎[††]　長名　保範[†††]
吉見　真聡[††††]　西川　由理[†††††]　天野　英晴[††††††]　舟橋　啓[††††††]　広井　賀子[††††††]
小栗　清[††]

Hideki YAMADA[†], Tomoya ISHIMORI[†], Tomonori OOYA[†], Yuichiro SHIBATA[††], Yasunori OSANA[†††], Masato YOSHIMI[††††], Yuri NISHIKAWA[†††††], Hideharu AMANO[††††††], Akira FUNAHASHI[††††††], Noriko HIROI[††††††], and Kiyoshi OGURI[††]

あらまし　FPGAを用いたシミュレーションシステムでは，回路をチップ上になるべく多く実現することによって並列効果を期待できることから，所望の回路を小規模に実現する技術が重要となる．本稿ではFPGAにおける生化学モデルのシミュレーションを対象に，シミュレーションターゲットに含まれる反応速度式の類似性に着目し，共通の演算回路群を自動的に結合し面積効率のよいパイプラインを合成する手法を提案する．また，この結合によるマルチプレクサの増大が招く周波数低下を抑制するために，反応速度式を類似のグループに分類する手法を提案する．実際の生化学モデルを用いた評価の結果，適切な反応速度式のクラスタリングを行えば，本手法を使わない場合に比べて約66%のハードウェア量で同等のシミュレーションを実行でき，その際の性能上のオーバヘッドは約20%であることが分かった．

1. はじめに

近年，生命現象をシステムとして解明しようという試みが盛んになっており，細胞全体のシミュレーションを対象としたE-Cell [1] やVirtual Cell [2] など数多くの生化学シミュレータが開発されている．計算機の性能は年々向上しているものの，シミュレーションの対象となる生化学システムも大規模化しており，シミュレーションに要する膨大な計算時間は依然として問題である．

我々が研究を進めているReCSiP(Reconfigurable Cell Simulation Platform) [3] は，SBML(Systems Biology Markup Language) [4] と呼ばれる生化学モデル記述言語で表現された常微分方程式に基づく反応モデルの求解算法をFPGA上の専用回路として実装することにより，高い演算スループットを実現するシミュレーションシステムである．FPGAを用いたシミュレーションシステムでは，回路をチップ上になるべく多く実現することによって，より大規模なモデルへの対応や並列処理効果を期待できることから，所望の回路を小規模に実現する技術が重要となる．本稿では，シミュレーションの対象となる生化学モデルに用いられる複数の反応速度式について，各々のデータフローグラフに共通に現れる演算群を自動的に抽出し，これを共有化したパイプラインを自動合成することで回路規模を削減する手法を提案する．また，この結合によるマルチプレクサの増大が招く周波数低下を抑制するために，反応速度式を類似のグループに分類する手法を提案する．

我々はこれまでにもReCSiPにおけるハードウェア量削減技術について検討を行ってきた．文献 [5] では，SBML定義済み関数 [4] と呼ばれる頻繁に使用される反応速度式に関して，手動で類似の反応速度式毎に演算器を共有化したハードウェアライブラリを実装した．これにより，回路面積を大きく削減できることが分かったが，利用可能な反応速度式が予め固定されていたため，モデルの汎用性の点で問題があった．文献 [6] では，任意の2つの反応速度式において，それぞれに共通な演算部分を自動的に見つけ出し，それらの演算を共通化する手法を提案した．しかしながら，この手法では反応速度式を3個以上共通化することはできなかった．

一方，共通演算部分の共有化による回路削減法の研究は，自動設計技術の分野において古くから行われてきているが，近年もリコンフィギャラブルシステムの持つ特性を活かした新しい手法が盛んに提案されている．例えば文献 [7] では，FPGAへ

[†] 長崎大学大学院生産科学研究科
[††] 長崎大学工学部
[†††] 成蹊大学理工学部
[††††] 同志社大学理工学部
[†††††] 慶應義塾大学大学院理工学研究科
[††††††] 慶應義塾大学理工学部

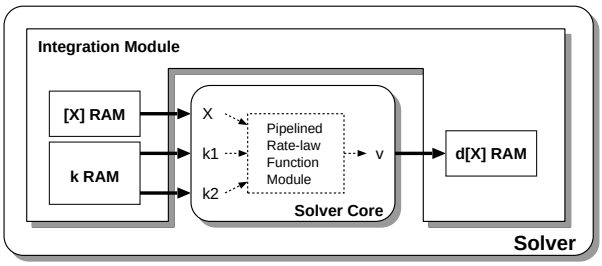

図 1　Solver モジュールの構成

のテクノロジマッピング時に，同形部分グラフを抽出し共通部分をまとめることで，回路面積を削減する手法が提案されている．また，文献 [8] では高位合成時にデータパス中の演算器を共有する新しい手法が提案されている．しかしながら，いずれの手法も共有するデータパスがパイプライン化されることは考慮しておらず，また，どのデータパス間で共有化を試みるべきかは既知であることが前提であり，この点で本稿で提案する手法と異なっている．

本稿の構成は次のとおりである．続く **2.** では生化学シミュレータ ReCSiP の構成を紹介する．次に，**3.** で提案手法の概要と処理の流れについて述べ，**4.** では生化学モデル中の反応速度式を類似度に従ってクラスタリングする手法について述べる．**5.** では反応速度式を表す複数のデータフローグラフから共通の演算群を共有化し結合する手法を示し，**6.** で本手法の評価結果を示しその有効性を考察する．最後に **7.** で本稿をまとめる．

2.　ReCSiP

FPGA を用いた生化学シミュレータ ReCSiP は，反応モデルの常微分方程式を解く Solver モジュール (図 1) が多数，スイッチを介して接続された構成を持つ．Solver は，反応速度を計算するための Solver Core と，数値積分を行う Integration Module の 2 つのモジュールから構成される．Solver Core は IEEE-754 に準拠した複数の単精度浮動小数点演算器とシフトレジスタ群から構成され，静的にスケジューリングされた演算パイプライン構造を持つ．

Solver Core における反応速度の計算には，反応に係わる複数の物質の濃度や反応速度定数が入力データとして使われる．Solver Core には，これらを入力するために，濃度変数用入力ポートが 1 ポート (図 1 中の X)，反応速度定数用入力ポートが 2 ポート (k1，k2) 用意されている．出力ポートは 1 ポートのみ (v) で，計算された反応速度が出力される．反応速度式ごとに Solver Core の構成は異なる．

反応速度式によっては，計算に必要な入力データ数が Solver Core の入力ポート数を越えることがある．この場合，すべての入力データを 1 クロックサイクルで Solver Core に投入することはできない．入力濃度変数の数を N_x，反応速度定数の数を N_k とすれば，データの投入には $P = \max(N_x, \lceil N_k/2 \rceil)$ クロックサイクル要することになる．

$P = 1$ のときには，十分な数の演算器を用意すればパイプラインが有効に機能し，1 クロックサイクルごとに新しい反応速度を計算できる．一方，$P \geqq 2$ では，P クロックサイクルあたり 1 回しか計算できないため，演算器にアイドルサイクルが生じる．そこで，同時に行われない演算について演算器を共有させることで，演算器数を削減できる．本稿では，この P のことをパイプラインピッチと呼ぶ．

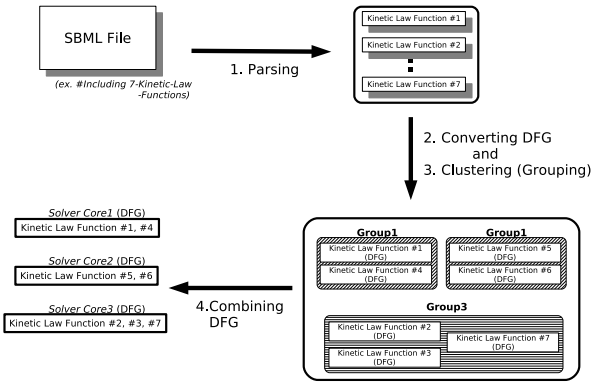

図 2　提案する処理のフロー

Solver では，同時に複数の種類の反応速度式を計算しない構造となっている．しかし，複数の種類の反応速度式を単一の Solver Core に実装しておき，必要に応じて，内部で反応速度式を切り替えて使用することが可能である．

3.　提案手法の流れ

本稿では，生化学モデル記述言語 SBML [4] で与えられたシミュレーションターゲット内の反応速度式について，類似の式ごとにグループを作り，それらのデータフローグラフ (DFG) 間の同形部分グラフをまとめることにより共通部分の回路面積を減らす手法を提案する．以下，本手法のことを DFG の結合と呼ぶ．

提案する処理のフローは，以下の 4 ステップである (図 2)．

（1） 与えられた SBML ファイルから反応速度式を抽出

（2） それぞれの反応速度式について DFG を生成

（3） 類似度に従って DFG を複数のクラスタに分類（クラスタリング）

（4） それぞれのクラスタごとに DFG を結合

結合された DFG は，演算やデータ投入のスケジューリング [9] を行った後，最終的には Solver Core 内部のパイプライン構造に変換される．

DFG の結合について，具体例を挙げて説明する．SBML 定義済み関数 [10] に含まれる反応速度式 UMI と UUCR はそれぞれ以下の式で表される．

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

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

上記の 2 つの DFG を図 3 に示す．網掛けで示した部分が 2 つの式の共通部分である．この共通部分の結合を行うと，図 4 の DFG が生成される．つまり，共通部分の回路をまとめ，異なる部分をマルチプレクサで接続する．また，3 個以上の DFG を

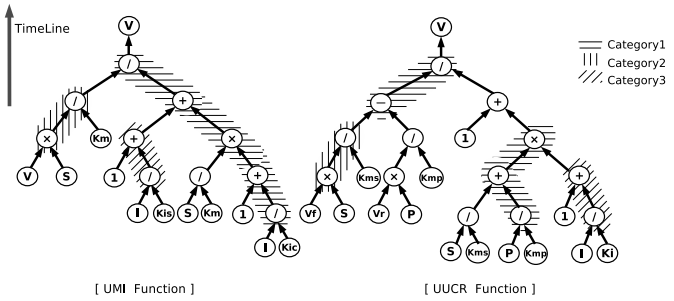

図 3　共通部分を抽出したデータフローグラフ

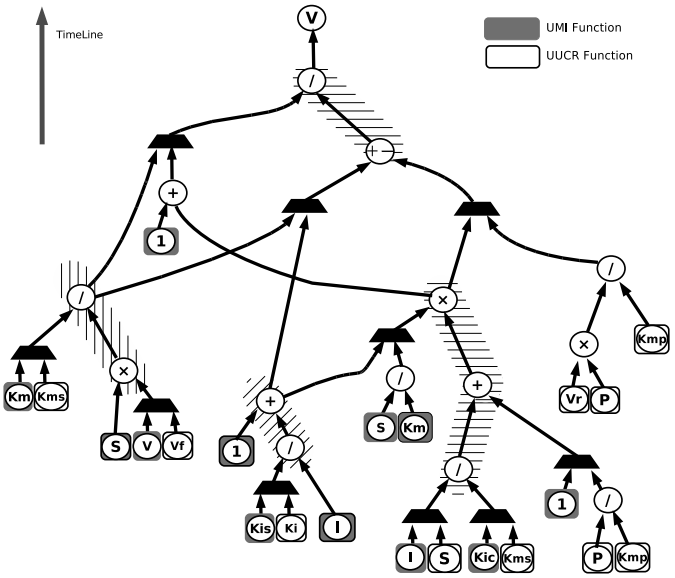

図 4　結合したデータフローグラフ

結合する際には，まず 2 つの DFG の結合を行い，その後，結合されたグラフに 1 つずつ DFG を追加結合する方法をとる．

4. DFG のクラスタリング

前章に述べた方法を用いれば，シミュレーションターゲット内に現れるすべての反応速度式を結合することもできるが，マルチプレクサ数の大きな増加を招くことから，Solver Core パイプラインの動作周波数の大幅な低下が容易に予想される．実行性能の低下をできるだけ抑制しながらハードウェア量を削減するには，使用されるすべての反応速度式を結合するのではなく，いくつかの反応速度式のグループを作り，それらのグループごとに結合を行うのが現実的である．このとき，類似した構造を持つ DFG の組合せごとにグループ化 (クラスタリング) して結合することが有効と考えられる (図 2 のステップ 3)．そこで，k 平均法アルゴリズムを用いてグラフ規模によるクラスタリングと最大同形部分グラフによるクラスタリングの 2 種類のアプローチを提案し，その効果を比較する．以降，それぞれの手法について順に説明する．2 つのアプローチを用いたときの比較評価については **6.** で議論する．

4.1　グラフの規模によるクラスタリング

1 つ目のアプローチは，k 平均法アルゴリズム [11] を用いて，同程度の規模の DFG を持つ反応速度式が同じグループになるようにクラスタリングを行うというものである．これは，規模が同程度の反応速度式を互いに結合することによって，Solver Core の実行性能の低下をできるだけ抑制するのがねらいである．

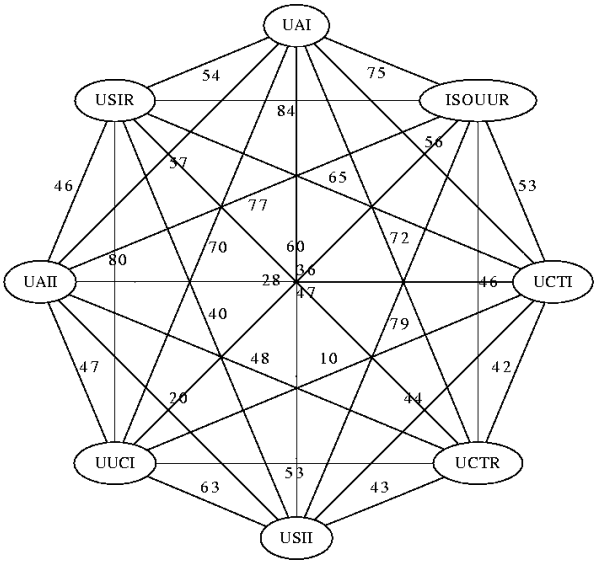

図 5　DFG 間の類似距離を表記した連合グラフの例

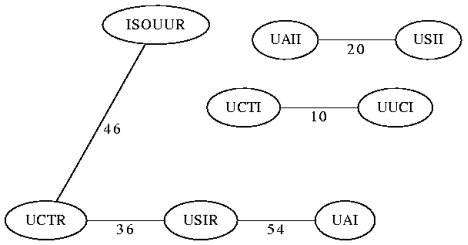

図 6　図 5 の DFG 群を 3 つのクラスタに分類した例

ここでは，DFG のノード数と高さを分類の評価に用いる．与えられた n 個の DFG を $g_i, i = 1, 2, \ldots, n$ とし，これを k 個のクラスタ $S_j, j = 1, 2, \ldots, k$ に分類する．具体的には，DFG g_i のノード数を n_{g_i}，高さを h_{g_i} で表し，$c_j \in S_j$ をクラスタ S_j の中心としたとき，

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

を最小とするように k 平均法を用いてクラスタリングを行う．その後，各クラスタごとに前章に述べた手法により反応速度式の結合を行う．

4.2　最大同形部分グラフによるクラスタリング

2 つ目のアプローチでは，グラフの類似度をより直接的に分類に反映させるために，最大同形部分グラフを用いる．今回は，2 つの DFG g_i と g_j の類似距離 d_{ij} を以下のように定める．

直感的には，DFG 全体に占める最大同形部分グラフの割合が大きいほど，2 つの DFG の類似度は高いと考えられる．そこで，DFG g_i，g_j のノード数を n_{g_i}，n_{g_j} とし，g_i と g_j 最大同形部分グラフのノード数を $\mathrm{MCS}(g_i, g_j)$ としたとき，n_{g_i} および n_{g_j} に占める $\mathrm{MCS}(g_i, g_j)$ の割合の平均を考える．ここで，d_{ij} のとりうる値を $0 \leqq d_{ij} \leqq 100$ とし，0 に近いほど 2 つの

DFG が類似していることを表すようにすれば，

$$d_{ij} = 100\left(1 - \frac{1}{2}\left(\frac{\mathrm{MCS}(g_i, g_j)}{n_{g_i}} + \frac{\mathrm{MCS}(g_i, g_j)}{n_{g_j}}\right)\right)$$
$$= 100 - 50\left(\frac{\mathrm{MCS}(g_i, g_j)}{n_{g_i}} + \frac{\mathrm{MCS}(g_i, g_j)}{n_{g_j}}\right) \quad (4)$$

と書ける．

この d_{ij} の値を結合の候補となる全ての DFG のペアに対して求め，図 5 に示すような連合グラフを求める．このグラフでは各ノードが DFG を表し，ノード間のエッジの重みがそのノード間の類似距離を示している．この後，各々のノードについて，最小の d_{ij} を持つエッジだけを残し，それ以外のエッジを除外する．このときに得られる連結部分グラフがクラスタとなる．要求されたクラスタ数 k よりも得られたクラスタ数が多い場合には，クラスタ数が k になるまで消去したエッジを類似距離の昇順に復元する，もし，得られたクラスタ数が k よりも少ない場合には，類似距離の降順にエッジをさらに消去する．以上の手順により，図 5 に示す連合グラフから 3 つのクラスタを得た結果を図 6 に示す．

5. 同形部分グラフの抽出

各クラスタに分類された DFG を結合するには，まず同形部分グラフを抽出する必要がある．本稿では，非連結な同形部分グラフを抽出する方法と，連結な同形部分グラフを抽出する方法の 2 種を実装し比較する．なお，以降の説明では i 番目の DFG g_i の j 番目のノードを $v_i(j)$ のように表記する．

5.1 非連結な同形部分グラフの抽出

非連結な同形部分グラフを探索するアルゴリズムを図 7，図 8 に示す．ここでは同形部分ブラフを抽出するために全てのノードの組合せを見ている．これは，生化学モデルに含まれる反応速度式の DFG の規模に鑑みて，単純な全探索でも現実的な計算時間で済むとの判断による．2 つのグラフ g_1 および g_2 の間の，あるノードの組合せ $(v_1(i), v_2(j))$ を引数として，COMMON_GRAPH_SET 関数を呼び出すことで，$(v_1(i), v_2(j))$ をルートノードとした同形部分グラフを得る．そして，この同形部分グラフのノード数が THRESHOLD を越えるもののみ，COMBINE_LIST 関数でリストに登録する．THRESHOLD は共有化の候補となる同形部分グラフの最小粒度を設定する役割を持つ．この値が小さいほど同形部分グラフの候補数は増えるが，共有化のオーバヘッドは相対的には増加すると予想される．ここにもトレードオフポイントが存在すると考えられるが，今回の実装ではもっとも細粒度のときの効果を評価するため THRESHOLD の値を 2 とした．ここで，登録される同形部分グラフのことをカテゴリ，登録されたカテゴリの集合をカテゴリセットと呼ぶことにする．図 8 での IsCommonNode 関数は，対応するノードが共有化できるかどうかを判定する関数であり，実際には，$v_1(i)$ と $v_2(j)$ が同じ演算器であるかを判定する (ただし，加算と減算は同値とみなす)．もし，共有化可能であれば，既に list に登録したノード以外の組合せのみ list に登録する．この後，$v_1(i)$ および $v_2(j)$ の子ノードを幅優先探索で辿っていき，終端まで繰り返し処理をさせる．最終的に，list に $v_1(i)$ と $v_2(j)$ をルートノードとした同形部分グラフのリストを得る．

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

図 7　非連結型同形部分グラフ探索アルゴリズム

```
INPUT: i, j, graph1, graph2
queue.push(i, j);
while(!queue.empty()){
  (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;
```

図 8　非連結型同形部分グラフ用 COMMON_GRAPH_SET アルゴリズム

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

図 9　連結型同形部分グラフ探索アルゴリズム

5.2 連結な同形部分グラフの抽出

連結な同形部分グラフを抽出するアルゴリズムを図 9 及び図 10 に示す．図 9 は，前節で示した図 7 と基本は同じである．IsCommonNode 関数も，図 7 のアルゴリズムと同様，対応するノードが共有化できるかを判定する関数である．common_node_chunk は，同形部分グラフを格納している．mcs は，それまでの探索で抽出された中で最大の同形部分グラフを格納している．もし，mcs のグラフのノード数が現在の common_node_chunk のノード数より少なければ，値が更新される．IsExtend 関数は，現在のノードの組合せを common_node_chunk に追加できるか否かを判定する関数である．ここでは，接続関係が連結である場合のみ true を返す．getNextPair 関数で，次のノード間の組合せを得る．

5.3 結合する同形部分グラフの選択

続いて，前述の手順で得られたカテゴリセット (結合の候補

```
INPUT: i, j, graph1, graph2, common_node_chunk
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()>mcs.size()){
         mcs=common_node_chunk;
      }
  }
}
while(isNextPair(i, j)){
  (next_i, next_j)=getNextPair(i, j);
  if(plunningCondition(next_i, next_j,
                              graph1, graph2)){
     mcs=COMMON_GRAPH_SET(next_i, next_j, graph1,
                                          graph2);
  }
}
return mcs;
```

図 10　連結型同形部分グラフ用 COMMON_GRAPH_SET アルゴリズム

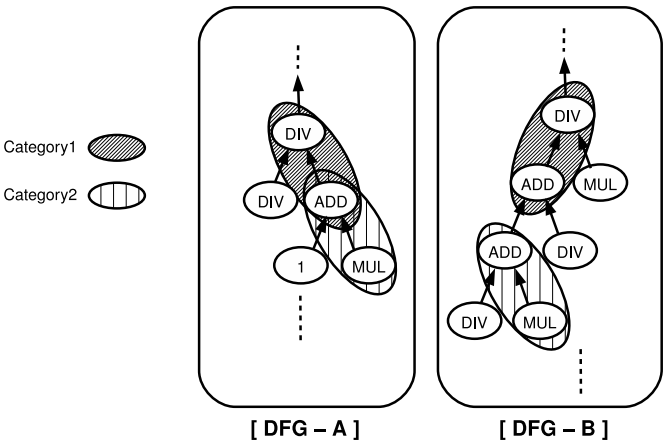

図 11　互いに重なりを持つ同形部分グラフの例

となる同形部分グラフの集合) の中から，実際に結合するカテゴリの組合せを選択する手法について述べる．

最終的なハードウェアコスト削減の観点からは，できるだけ多くのノードを包含するようなカテゴリの組合せが望ましい．しかし，パイプラインを構成するために以下 2 つの条件を満たさなければならない．

（1） 選択する同形部分グラフは，重複部分を持たないこと

（2） 結合の結果，循環グラフが生成されないこと

まず，(1) について，具体例を挙げて説明する．図 11 は，2 つのデータフローグラフ DFG-A および DFG-B の間に，結合の候補として 2 つの同形部分グラフ Category 1 および Category 2 が抽出された例を示している．しかしながら，これら 2 つのカテゴリは重なりを持つ．このため，これらを双方とも選択して結合した場合，DFG-A の ADD 演算子と，DFG-B の ADD 演算子の対応関係からパイプラインの制御上の問題が生じる．

(2) の制約を満たさず循環グラフが生成されるのは，結合する同形部分グラフの演算順序の関係が互いに交差するときである．このような例を，図 12 に示す．結合の結果，Category 1 と Category 2 に相互依存関係が生じ，パイプライン化を妨げている．

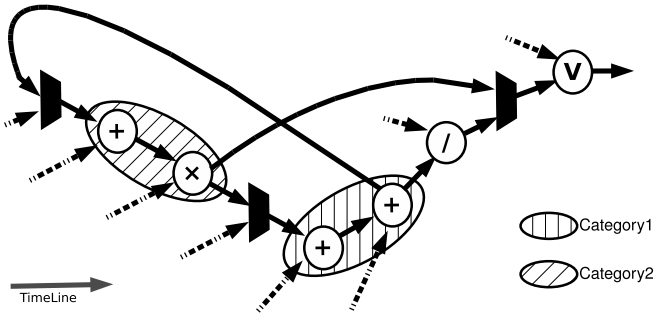

図 12　循環グラフを生成する例

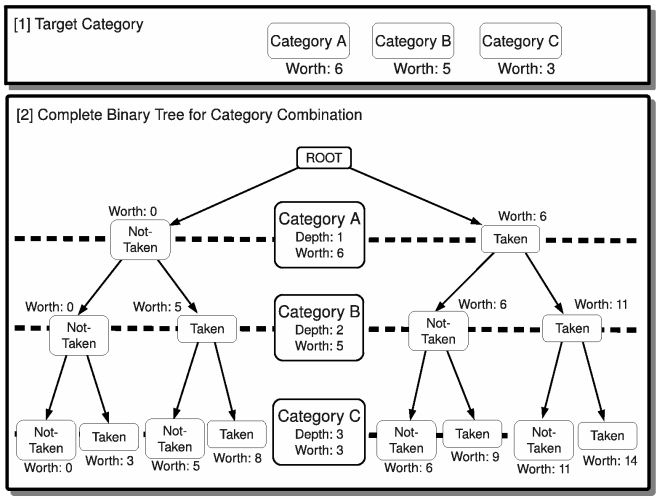

図 13　二分決定木による組合せの表現

カテゴリセットの中から，より多くのノードを包含するようなカテゴリの組合せを選択する問題は，選択したカテゴリに含まれる全ノード数を目的関数 (価値) としたナップザック問題に類似しているが，同時に選択できるカテゴリに関して前述の制約がある点や，選択できるカテゴリの総数に関しては制約がない点などが異なっている．特に後者の性質から，探索すべき組合せの数は膨大となる．

そこで，図 13 に示すような二分決定木を用いて，なるべく多くのノード数を被覆するカテゴリの組合せを短時間に選択する手法を提案する．まず，カテゴリセット内のカテゴリをノード数の降順に並べ，二分決定木のルートノード側から木の階数に対応させる．例えば，図 13 の例では 3 つのカテゴリ (Category A，Category B，Category C) を順にルートノード側から対応させている．すなわち，ルートノードから順に，Category A を選択するならば右側のエッジ，選択しないならば左側のエッジというように進み，リーフノードに達したところで選択したカテゴリの組合せに含まれる総ノード数が得られる．この例では示していないが，前述の制約により同時に選択できない組合せについては対応する二分決定木のエッジを除去しておく．また，探索時間を短縮するために，二部決定木においてどの深さまでを探索するかの閾値をあらかじめ定めておき，その深さまでの探索で得られたうちでもっともノード数の多いカテゴリの組合せを選択する．この探索を打ち切る深さ閾値に

表 1　使用した演算器ライブラリの諸元

演算	レイテンシ (サイクル)	スライス数	最大動作周波数 (MHz)
加算	6	605	120.49
乗算	8	230	271.08
除算	17	1684	146.98
開平	23	1061	151.09
対数	33	2386	115.87
冪乗	81	5406	115.54

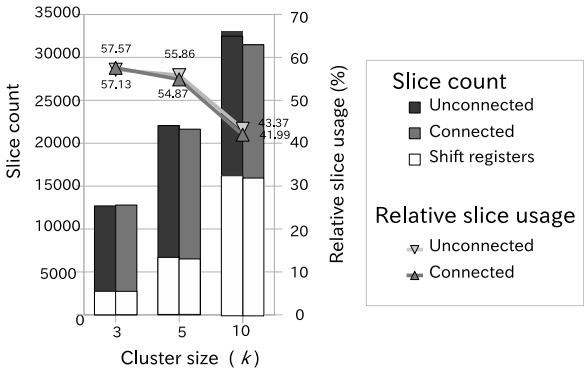

図 14　スライス使用率

よる影響についても次章で評価する．

6. 評価と考察

前章までに述べたように，提案手法にはいくつかの選択肢があるため，まず，それらをすべて実装した上で以下の各観点について比較評価を行った．

- 始めに抽出する同形部分グラフは，非連結がよいか連結がよいか
- 2 分決定木を用いた同形部分グラフの組合せ探索を打切る深さ閾値は，結合の結果にどの程度影響するか
- DFG のクラスタリングは，グラフの規模によるものがよいか，最大同形部分グラフによるものがよいか

提案手法の処理系を C++(gcc4.1.2 +O3) で実装し，SBML 定義済み関数の 32 個の反応速度式 [4] を用いて評価した．また，結合した DFG に対して演算スケジューリングや演算器共有化処理 [9] を行い演算パイプラインの RTL 記述を Verilog-HDL で生成するバックエンドも実装し，これを ReCSiP ボードに搭載された Xilinx 社製 XC2VP70-5 FPGA をターゲットデバイスとして実装した．演算器には表 1 に示すパイプライン型のものを使用した．論理合成と配置配線には Xilinx 社製 ISE-10.3i を用いた．

6.1　抽出する同形部分グラフの連結性の影響

まず，抽出する同形部分グラフの連結性がどの程度結合結果に影響を与えるかを評価した．クラスタリング手法の影響を排除して考察するために，SBML 定義済み関数からランダムに k 個の関数を選択し結合させる実験を，$k = 3, 5, 10$ のそれぞれについて 20 回行った．このとき，2 分決定木による探索の打切り深さは 10 に固定した．

これらの結果を手法別にプロットしたものを図 14 に示す．棒グラフに結合の結果合成された Solver Core の使用スライス数を 20 回の試行の平均値で示している．また，使用スライス数のうち，演算ロジックが占める割合と，シフトレジスタが占める割合の内訳も合わせて示している．結合する関数の数 (k) が 5 と 10 の場合には，連結な同形部分グラフを用いた方が，非連結な同形部分グラフを用いるよりも良好な結果が得られることが分かる．一方，$k = 3$ の場合は，非連結な同形部分グラフを用いた方がわずかに使用スライス数が少なくて済んでいる．

非連結な同形部分グラフは連結な同形部分グラフに比べて，部分グラフ 1 つあたりのノード数は多くなる傾向がある．このため，$k = 3$ のときには，非連結な同形部分グラフを用いて結合することにより，より多くの演算器が共有されたと考えられる．逆に，同形部分グラフが大きくなると，グラフの重なりや相互依存による制約違反の可能性が生じやすくなる．このため，非連結な同形部分グラフを用いる手法は，結合させる関数が増えると，かえって結合の効果を損ねたと考えられる．

図 14 の折れ線グラフは，結合を行った際の使用スライス数を，結合を行わなかった場合の使用スライス数を 1 として百分率で表したものである．このグラフからも分かるように，10 個の関数を結合させた場合には，結合しなかった場合と比べて約 40%程度のハードウェア量で Solver Core を実現できている．ただし，10 個の関数を結合させた場合には，全体のハードウェア量のほぼ半分がシフトレジスタによって占められており，より多くの関数を結合させて面積削減を図るにはシフトレジスタ使用量の最適化についても考慮しなければならないことが示唆されている．

6.2　2 分決定木の探索打切り深さ閾値の影響

次に 2 分決定木の探索打切り深さ閾値の影響を評価するために，深さ閾値を 5，10，20，30，40 と変化させて同様の実験を行った．結合する関数の個数 k を 3，5，10 としたときの結果を，それぞれ図 15〜図 17 に示す．グラフの縦軸は，結合した結果の使用スライス数を，結合しなかったときの使用スライス数を 1 として百分率で表したものであり，それぞれ 5 回の試行の平均値を示している．

これらの結果から分かるように，深さ閾値は 20〜30 程度より大きくしても面積削減効果はそれほど変わらない．これは探索が深くなるほど同形部分グラフ選択の制約が生じやすくなるためと考えられる．したがって，深さ閾値を 20〜30 程度に設定しそこで探索を打切ることにより，結合の効果をほとんど損なうことなく探索空間を削減し結合処理を高速化できることが分かる．また，$k = 5$ と $k = 10$ の場合には，すべての深さ閾値に対して連結な同形部分グラフを用いる手法がわずかに良い結果を示し，同形部分グラフの連結性の優劣は探索を打切る深さ閾値に影響されないことも明かになった．

6.3　クラスタリング手法による影響

4. で述べた DFG のクラスタリング手法が結合結果に与える影響を評価するために，32 個の SBML 定義済み関数を 5 つのクラスタに分類し結合させる実験を，グラフ規模による分類と最大同形部分グラフによる分類の両方で行った．ここでは，結

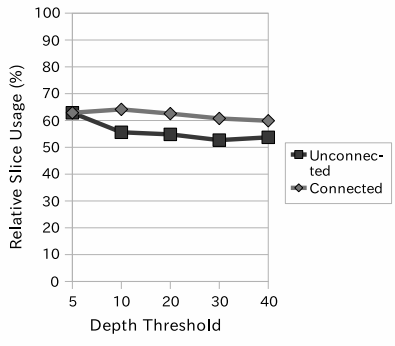

図 15　相対使用スライス数 ($k = 3$)

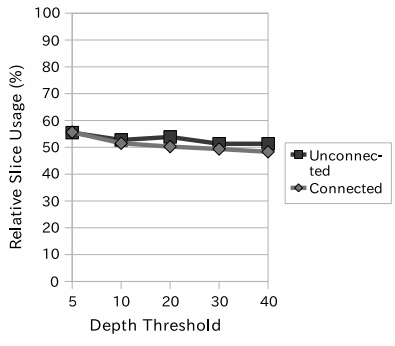

図 16　相対使用スライス数 ($k = 5$)

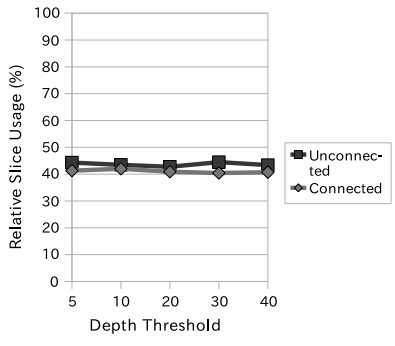

図 17　相対使用スライス数 ($k = 10$)

表 2　クラスタリング手法による結合結果への影響

クラスタリング手法	グラフ規模	最大同形部分グラフ
レイテンシ悪化率 (%)	19.16	10.87
動作周波数低下率 (%)	8.33	7.43
相対スライス数 (%)	51.11	39.27

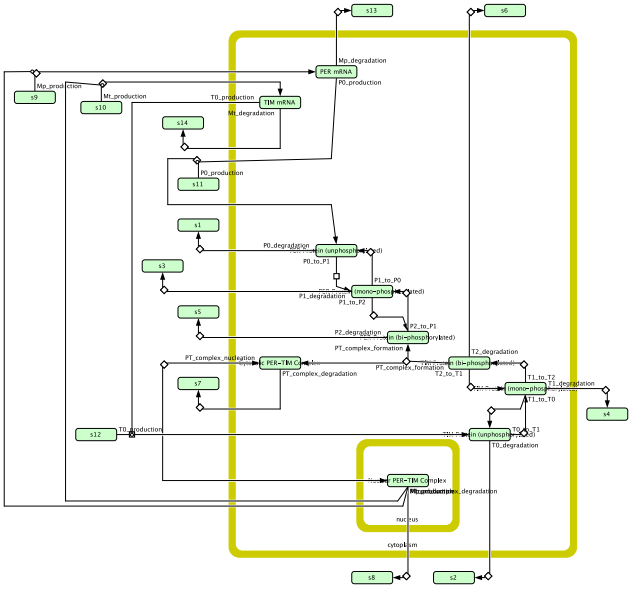

図 18　評価に用いた生化学モデル

合の性能面でのオーバヘッドとして，結合しなかった場合と比較した Solver Core のレイテンシの悪化率および動作周波数の低下率を評価した．また，面積削減の効果として，結合せずに実装したときの使用スライス数を 1 とした相対使用スライス数も評価し，これらの結果を表 2 にまとめた．

これらの結果が示している通り，結合に起因する Solver Core の性能低下と，面積削減効果の両面において，最大同形部分グラフを用いたクラスタリング手法に優位性が認められた．これは単に DFG の規模だけではなく，DFG の類似度をより詳細に考慮してどの DFG を結合すべきかを決定することの重要性を裏付けている．

6.4　生化学モデルを用いた評価

最後に実際の生化学モデルを用いて提案手法の効果を評価した．評価には，Leloup らのショウジョウバエ概日リズムのモデルを用いた [12]．このモデルは，ショウジョウバエ概日リズムを司る PER および TIM タンパクの周期的な発現変化を，転写段階での両者の複合体を介したネガティブフィードバック機構によって再現したものである（図 18）．このモデルには 7 種類の SBML 定義済み反応速度式が含まれ，これらを以下に示す 3 種類の方法でそれぞれ実装し，ReCSiP 上でシミュレーションを実行した．

（1） Not-Combined: 結合手法を用いずに，7 個の反応速度式全てについて独立の Solver Core を生成．

（2） All-Combined: 7 個の反応速度式全てを結合し，単一の Solver Core に共有化．

（3） Clustered: 7 個の反応速度式を 3 個のクラスタに分類し，3 個の Solver Core を生成．

このうち，All-Combined および Clustered については，連結型同形部分グラフの抽出を行い，2 分決定木の探索打切り深さ閾値は 30 とした．また，Clustered における分類手法には，最大同形部分グラフによる分類を用いた．

それぞれの実装について，シミュレーション回路全体の使用スライス数を図 19 に示す．結果から明らかなように，Clustered がこの中でもっとも良好な結果を示した．DFG を結合するレベルでは All-Combined がもっとも多くのノードを共有化できたが，DFG をパイプラインハードウェアに変換する演算スケジューリングと演算器割当ての際に，冪乗演算を効果的に共有化して削減できなかったことから，最終的には Clustered がもっとも少ないハードウェア量で実装できた．このことは，結合手法の効果は，単に DFG のレベルで多くのノードを共有するだけでは十分ではなく，バックエンドのスケジューリングの影響も受けることを示しており，本稿で議論している問題の複雑さを示している．シミュレーション回路全体では，結合を行わなかった時に比べて，All-Combined は 74.82%，Clustered では 65.73%のハードウェア量で同等のシミュレーションを実行できることが分かった．

次に，シミュレーション回路全体の動作周波数を図 20 に示す．結合によって Solver Core に切り替え用のマルチプレクサ等が挿入されるため，結合を行わない実装 (Not-Combined) に比べ，結合を行った実装 (All-Combined，Clustered) は，共に 15.38%の周波数低下を示した．また，シミュレーションスループット (1 秒間に計算できる反応の数) の評価結果を図 21 に示

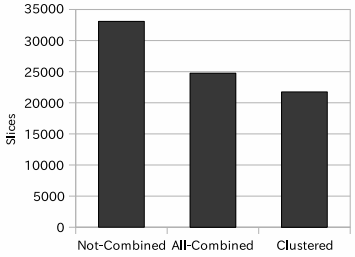

図 19　シミュレーション回路全体の使用スライス数

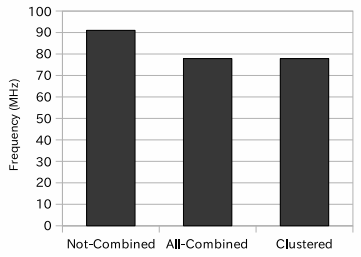

図 20　シミュレーション回路全体の動作周波数

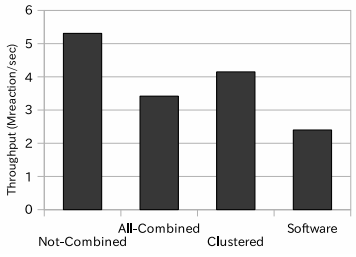

図 21　シミュレーションスループット (33MHz 動作時)

す．この結果は，使用した ReCSiP ボードに実装されているクロックジェネレータの関係から，33MHz の動作周波数で評価したものである．図 21 には，比較のため，3.8GHz の Pentium4 を搭載した Linux PC 上で同じモデルのシミュレーションをソフトウェア処理したときの結果も示している．結合手法を用いない Not-Combined のスループットは，ソフトウェアの 2.21 倍，All-Combined，Clustered は，それぞれ 1.43 倍，1.73 倍のスループットを示した．また図 20 に示した周波数で動作させた場合と比較すると，Not-Combined で 6.10 倍，All-Combined，Clustered は，それぞれ 3.33 倍，4.08 倍のスループットとなった．結合を用いたハードウェア量削減手法には，20%程度の性能上のオーバヘッドも存在するが，適切なクラスタリングを行えばハードウェア量削減効果は大きく，ソフトウェアに対する性能上の優位性も保たれることから，特に大規模な生化学モデルをターゲットとする際には有力な手法であると考えられる．

7. むすび

本稿では，生化学モデルに現れる任意の反応速度式群について，データフローグラフ上に現れる共通の演算回路を自動的に結合することで，FPGA を用いた生化学シミュレータの回路面積を削減する手法を提案した．また，マルチプレクサの増大による周波数低下を抑制するために，反応速度式の類似性に着目し結合すべき類似グループに分類する手法を提案した．

実際の生化学モデルを用いた評価の結果，適切な反応速度式のクラスタリングを行えば，本手法を使わない場合に比べて約 66%のハードウェア量で同等のシミュレーションを実行でき，その際の性能上のオーバヘッドは約 20%であることが分かった．

今後の課題としては，結合候補となる同形部分グラフの最適な最小ノード数の決定や評価結果より必要性が明かになったパイプラインスケジューリングも考慮に入れた DFG の結合手法の検討，結合する関数を増やした際に支配的になるシフトレジスタの回路面積の削減手法の検討などが挙げられる．

文　献

[1] M. Tomita, K. Hashimoto, K. Takahashi, T.S. Shimizu, Y. Matsuzaki, F. Miyoshi, K. Saito, S. Tanida, K. Yugi, J.C. Venter, and C.A. Hutchison, “E-CELL: software environment for whole cell simulation,” Bioinformatics, vol.15, pp.72–84, 1999.

[2] I.I. Moraru, J.C. Schaff, B.M. Slepchenko, and L.M. Loew, “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, M. Yoshimi, Y. Iwaoka, T. Kojima, Y. Nishikawa, A. Funahashi, N. Hiroi, Y. Shibata, N. Iwanaga, H. Kitano, and H. Amano, “ReCSiP: An FPGA-based general-purpose biochemical simulator,” Electronics and Communications in Japan, part 2, vol.90, no.7, pp.1–10, 2007.

[4] M. Hucka, A. Finney, B.J. Bornstein, S.M. Keating, B.E. Shapiro, J. Matthews, B.L. Kovitz, M.J. Schilstra, A. Funahashi, J.C. Doyle, and H. Kitano, “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.

[5] N. Iwanaga, Y. Shibata, M. Yoshimi, Y. Osana, Y. Iwaoka, T. Fukushima, H. Amano, A. Funahashi, N. Hiroi, H. Kitano, and K.Oguri, “Efficient scheduling of rate law functions for ODE-based multimodel biochemical simulation on an FPGA,” Proc. International Conference on Field-Programmable Logic and Applications, pp.666–669, 2005.

[6] H. Yamada, N. Iwanaga, Y. Shibata, Y. Osana, M. Yoshimi, Y. Iwaoka, Y. Nishikawa, T. Kojima, H. Amano, A. Funahashi, N. Hiroi, H. Kitano, and K. Oguri, “A combining technique of rate law functions for a cost-effective reconfigurable biological simulator,” Proc. International Conference on Field-Programmable Logic and Applications, pp.808–811, 2007.

[7] A.M. Smith, G.A. Constantinides, and P.Y.K. Cheung, “Fused-arithmetic unit generation for reconfigurable devices using common subgraph extraction,” Proc. International Conference on Field-Programmable Technology, pp.105–112, 2007.

[8] M. Fazlali, M.K. Fallah, M. Zolghadr, and A. Zakerolhosseini, “A new datapath merging method for reconfigurable system,” Proc. International Workshop on Reconfigurable Computing, pp.157–168, 2009.

[9] T. Ishimori, H. Yamada, Y. Shibata, Y. Osana, M. Yoshimi, Y. Nishikawa, H. Amano, A. Funahashi, N. Hiroi, and K. Oguri, “Pipeline scheduling with input port constraints for an fpga-based biochemical simulator,” Proc. International Workshop on Applied Reconfigurable Computing, pp.368–373, 2009.

[10] M. Hucka, A. Finney, H. Sauro, and H. Bolouri, “Systems biology markup language (SBML) level 1: Structures and facilities for basic model definitions,” 2003. http://sbml.org/Special/specifications/sbml-level-1/version-2/sbml-level-1-v2.pdf

[11] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” Proc. Berkeley Symposium on Mathematical Statistics and Probability, vol.1, pp.281–297, 1967.

[12] J.-C. Leloup and A. Goldbeter, “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.